Creating Audio Apps for watchOS

Session 504

Neil Desai, watchOS Frameworks Engineer
Native controls
Getting content
Local playback
Audio experience
Native controls
Getting content
Local playback
Audio experience
Native controls
Getting content
Local playback
Audio experience
Native controls
Getting content
Local playback
Audio experience
Native controls
Getting content
Local playback
Audio experience
Native Controls
Native Controls
Native Controls
Native Controls
Native Controls
Now playing view
Native Controls

Now playing view

Digital Crown controls volume
Native Controls
Now playing view
Digital Crown controls volume
Place in non-scrolling controller
Native Controls

Now playing view

Digital Crown controls volume

Place in non-scrolling controller

Automatically switches sources
Native Controls
Now playing view

Digital Crown controls volume
Place in non-scrolling controller
Automatically switches sources
Application tint color
Now Playing View - Displays Now Playing View.
Native Controls
Volume control

Controls iPhone or local volume
Native Controls

Volume control

Controls iPhone or local volume

Application tint color
Native Controls
Volume control

Controls iPhone or local volume

Application tint color
Native Controls
Volume control
Native Controls

Volume control
Native Controls

Volume control
Native Controls

Volume control
Getting Content
Getting Content
Getting Content
//User initiated download

@IBAction func userPressedDownload() {
    WKExtension.shared().isFrontmostTimeoutExtended = true
    let session = MyNSURLSessionDelegate.backgroundSession
    let task = session.downloadTask(with: contentURL)
    task.resume()
}

@IBAction func userPressedDownload() {
    WKExtension.shared().isFrontmostTimeoutExtended = true
    let session = MyNSURLSessionDelegate.backgroundSession
    let task = session.downloadTask(with: contentURL)
    task.resume()
}

// User initiated download
// User initiated download

@IBAction func userPressedDownload() {
    WKExtension.shared().isFrontmostTimeoutExtended = true
    let session = MyNSURLSessionDelegate.backgroundSession
    let task = session.downloadTask(with: contentURL)
    task.resume()
}
class DownloadManager: NSObject, URLSessionTaskDelegate, URLSessionDownloadDelegate {
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didWriteData bytesWritten: Int64, totalBytesWritten: Int64, totalBytesExpectedToWrite: Int64) {
        if totalBytesExpectedToWrite > 0 {
            let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
            // Update UI showing progress
        }
    }
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didFinishDownloadingTo location: URL) {
        // Manage file
        // Post notification alerting user of playback opportunity
    }
    func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: Error?) {
        // Clean up state, handle errors if there are any
    }
}
class DownloadManager : NSObject, URLSessionTaskDelegate, URLSessionDownloadDelegate {
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didWriteData bytesWritten: Int64, totalBytesWritten: Int64, totalBytesExpectedToWrite: Int64) {
        if totalBytesExpectedToWrite > 0 {
            let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
            // Update UI showing progress
        }
    }
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didFinishDownloadingTo location: URL) {
        // Manage file
        // Post notification alerting user of playback opportunity
    }
    func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: Error?) {
        // Clean up state, handle errors if there are any
    }
}
class DownloadManager : NSObject, URLSessionTaskDelegate, URLSessionDownloadDelegate {
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didWriteData bytesWritten: Int64, totalBytesWritten: Int64, totalBytesExpectedToWrite: Int64) {
        if totalBytesExpectedToWrite > 0 {
            let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
            // Update UI showing progress
        }
    }

    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didFinishDownloadingTo location: URL) {
        // Manage file
        // Post notification alerting user of playback opportunity
    }

    func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: Error?) {
        // Clean up state, handle errors if there are any
    }
}
class DownloadManager: NSObject, URLSessionTaskDelegate, URLSessionDownloadDelegate {
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didWriteData bytesWritten: Int64, totalBytesWritten: Int64, totalBytesExpectedToWrite: Int64) {
        if totalBytesExpectedToWrite > 0 {
            let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
            // Update UI showing progress
        }
    }

    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didFinishDownloadingTo location: URL) {
        // Manage file
        // Post notification alerting user of playback opportunity
    }

    func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: Error?) {
        // Clean up state, handle errors if there are any
    }
}
class DownloadManager : NSObject, URLSessionTaskDelegate, URLSessionDownloadDelegate {
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didWriteData bytesWritten: Int64, totalBytesWritten: Int64, totalBytesExpectedToWrite: Int64) {
        if totalBytesExpectedToWrite > 0 {
            let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
            // Update UI showing progress
        }
    }

    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didFinishDownloadingTo location: URL) {
        // Manage file
        // Post notification alerting user of playback opportunity
    }

    func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: Error?) {
        // Clean up state, handle errors if there are any
    }
}
Getting Content
Getting Content

WatchConnectivity
WatchConnectivity
WatchConnectivity

Transfer file API
WatchConnectivity

Transfer file API

Progress
// Progress property

open class WCSessionFileTransfer : NSObject {

    open var file: WCSessionFile { get }

    @available(iOS 12.0, *)
    open var progress: Progress { get }

    open var isTransferring: Bool { get }

    open func cancel()
}

open class WCSessionFileTransfer : NSObject {

    open var file: WCSessionFile { get }

    @available(iOS 12.0, *)
    open var progress: Progress { get }

    open var isTransferring: Bool { get }

    open func cancel()
}
// User initiated download

if let transfer: WCSessionFileTransfer = fileTransfer {
    transferProgressView.observedProgress = transfer.progress
}
// User initiated download

if let transfer: WCSessionFileTransfer = fileTransfer {
    transferProgressView.observedProgress = transfer.progress
}

// User initiated download

if let transfer: WCSessionFileTransfer = fileTransfer {
    transferProgressView.observedProgress = transfer.progress
}

SimpleWatchConnectivity

Sample Code
Which one do we use?
URLSession
URLSession

User initiated on Apple Watch

Use `waitsForConnectivity` instead of `SCNetworkReachability`
URLSession

User initiated on Apple Watch

Use `waitsForConnectivity` instead of `SCNetworkReachability`

Requests are proxied through iPhone, when in range
User initiated on Apple Watch

Use `waitsForConnectivity` instead of `SCNetworkReachability`

Requests are proxied through iPhone, when in range
WatchConnectivity
WatchConnectivity

Initiated on iPhone
WatchConnectivity

Initiated on iPhone

No need to request from your server again
Set Expectations
Set Expectations

Instruct your user
Set Expectations

Instruct your user

On the magnetic charger
Music syncs when Apple Watch is on its charger. Once it’s synced, music will be available on Apple Watch even when it’s out of range of your iPhone.

AUTOMATICALLY ADD
- **Heavy Rotation**: Playlists and albums you’re listening to
- **Favorites Mix**: Updated every Tuesday
- **Chill Mix**: Updated every Sunday
- **New Music Mix**: Updated every Friday

PLAYLISTS & ALBUMS
- Add Music...
Music syncs when Apple Watch is on its charger. Once it’s synced, music will be available on Apple Watch even when it’s out of range of your iPhone.
Local Playback
WatchKit
WatchKit

presentMediaPlayerController(with: options:completion:)
WatchKit

presentMediaPlayerController(with: options:completion:)

WKAudioFileQueuePlayer
WatchKit

`presentMediaPlayerController(with: options:completion:)`

`WKAudioFileQueuePlayer`

- `WKAudioFilePlayerItem`
Playback
WKAudioFileQueuePlayer
AVFoundation

AVAudioPlayer
AVFoundation

AVAudioPlayer

AVAudioEngine
AVFoundation

AVAudioPlayer

AVAudioEngine

Playback

• Background run mode

• Foreground, screen on
Audio Playback Background Mode
Background Playback
Background Playback

AVFoundation

NEW
Background Playback

AVFoundation

Route picking
Background Playback

AVFoundation
Route picking
MediaPlayer.framework
Background Playback

AVFoundation

Route picking

MediaPlayer.framework

Restricted to Bluetooth routes just like Music, Podcasts, and Radio
Background Playback
Background Playback

Audio Info.plist key
Background Playback

Audio Info.plist key → Set up your session
Background Playback

Audio Info.plist key → Set up your session → Route picking
Background Playback

1. Audio Info.plist key
2. Set up your session
3. Route picking
4. Play!
Background Modes

Modes: □ Audio
□ Location updates
□ Workout processing

Steps: □ Add the Required Background Modes key to your info plist file
Local Playback
Local Playback

Set `routeSharingPolicy` to `longForm` on AVAudioSession
Local Playback

Set `routeSharingPolicy` to `longForm` on `AVAudioSession`

Call new `activate(withOptions:completion:)` method on `AVAudioSession`
Local Playback

Set `routeSharingPolicy` to `longForm` on `AVAudioSession`

Call new `activate(withOptions:completion:)` method on `AVAudioSession`

• On completion, call `play()`
Route Picker
Route Picker

Async
activate()
Route Picker

Async
activate()
Route Picker

Async activate()

completion()
func setupAudioSession() throws {
  try AVAudioSession.sharedInstance().setCategory(
    AVAudioSessionCategoryPlayback,
    mode: AVAudioSessionModeDefault,
    routeSharingPolicy: .longForm,
    options: []
  )
}

func setupAudioSession() throws {
    try AVAudioSession.sharedInstance().setCategory(
        AVAudioSessionCategoryPlayback,
        mode: AVAudioSessionModeDefault,
        routeSharingPolicy: .longForm,
        options: []
    )
}
// Activate session, display route picker, play!

@IBAction func userPressedPlay() {
    AVAudioSession.sharedInstance().activate(options: []) {
        (error: Error?) in
        if let error = error {
            print(error)
        } else {
            self.player?.play()
        }
    }
}
// Activate session, display route picker, play!

@IBAction func userPressedPlay() {
    AVAudioSession.sharedInstance().activate(options: []) { (error: Error?) in
        if let error = error {
            print(error)
        } else {
            self.player?.play()
        }
    }
}
// Activate session, display route picker, play!

@IBAction func userPressedPlay() {
    AVAudioSession.sharedInstance().activate(options: []) { (error: Error?) in
        if let error = error {
            print(error)
        } else {
            self.player?.play()
        }
    }
}
Background Playback
Routes
Background Playback

Routes

Apple wireless chip, W1
Background Playback

Routes

Apple wireless chip, W1

Bluetooth headphones
Background Playback
Route picker

Route picker details
Background Playback

Route picker

Route picker details

• If user has an active route, the route picker will select that automatically on your behalf
Background Playback

Route picker

Route picker details

• If user has an active route, the route picker will select that automatically on your behalf

• Takes active W1 route from iPhone
Background Playback
Route picker

Route picker details
• If user has an active route, the route picker will select that automatically on your behalf
• Takes active W1 route from iPhone
  - Unless iPhone has more priority
Background Playback
Route picker

Route picker details

• If user has an active route, the route picker will select that automatically on your behalf
• Takes active W1 route from iPhone
  - Unless iPhone has more priority
• If no active routes, route picker will appear
Background Playback

Playback

• AVAudioPlayer
• AVAudioEngine

Formats supported

AAC-LC, AAC-ELD, HE-AAC, HE-AACv2, MP3 (decoding only), Opus
Background Playback

Power

Only play audio when necessary

AVAudioEngine

• autoShutdownEnabled on by default
Background Playback
MediaPlayer.framework
Background Playback
MediaPlayer.framework

Provide Now Playing information
Background Playback
MediaPlayer.framework

Provide Now Playing information

Now Playing UI will update
Background Playback
MediaPlayer.framework

Provide Now Playing information

Now Playing UI will update

Handle events
// Providing Now Playing information

fileprivate let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()

var nowPlayingInfo = [String: Any]()
let image = UIImage(data: artworkData) ?? UIImage()
let artwork = MPMediaItemArtwork(boundsSize: image.size, requestHandler: { (_, ) -> UIImage in return image })

nowPlayingInfo[MPMediaItemPropertyTitle] = title
nowPlayingInfo[MPMediaItemPropertyAlbumTitle] = album
nowPlayingInfo[MPMediaItemPropertyArtwork] = artwork

nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
// Providing Now Playing information

fileprivate let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()

var nowPlayingInfo = [String: Any]()
let image = UIImage(data: artworkData) ?? UIImage()
let artwork = MPMediaItemArtwork(boundsSize: image.size, requestHandler: { (_) -> UIImage in
    return image
})

nowPlayingInfo[MPMediaItemPropertyTitle] = title
nowPlayingInfo[MPMediaItemPropertyAlbumTitle] = album
nowPlayingInfo[MPMediaItemPropertyArtwork] = artwork

nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
// Providing Now Playing information

fileprivate let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()

var nowPlayingInfo = [String: Any]()
let image = UIImage(data: artworkData) ?? UIImage()

let artwork = MPMediaItemArtwork(boundsSize: image.size, requestHandler: { (_) -> UIImage in
    return image
})

nowPlayingInfo[MPMediaItemPropertyTitle] = title
nowPlayingInfo[MPMediaItemPropertyAlbumTitle] = album
nowPlayingInfo[MPMediaItemPropertyArtwork] = artwork

nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
// Providing Now Playing information

fileprivate let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()

var nowPlayingInfo = [String: Any]()

let image = UIImage(data: artworkData) ?? UIImage()

let artwork = MPMediaItemArtwork(boundsSize: image.size, requestHandler: { (_) -> UIImage in
    return image
})

nowPlayingInfo[MPMediaItemPropertyTitle] = title
nowPlayingInfo[MPMediaItemPropertyAlbumTitle] = album
nowPlayingInfo[MPMediaItemPropertyArtwork] = artwork

nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
// Providing Now Playing information

fileprivate let nowPlayingInfoCenter = MPNowPlayingInfoCenter.default()

var nowPlayingInfo = [String: Any]()
let image = UIImage(data: artworkData) ?? UIImage()
let artwork = MPMediaItemArtwork(boundsSize: image.size, requestHandler: { (Imagen in
  return image
})

nowPlayingInfo[MPMediaItemPropertyTitle] = title
nowPlayingInfo[MPMediaItemPropertyAlbumTitle] = album
nowPlayingInfo[MPMediaItemPropertyArtwork] = artwork

nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
Lights in the Sky
Nine Inch Nails—The
Background Playback
Media remote

Use MediaPlayer.framework

Handle commands however you wish
@objc class RemoteCommandManager: NSObject {
    // Reference of `MPRemoteCommandCenter` used to configure and
    // setup remote control events in the application.
    fileprivate let remoteCommandCenter = MPRemoteCommandCenter.shared()
}
//Setup

@objc class RemoteCommandManager: NSObject {
    // Reference of `MPRemoteCommandCenter` used to configure and
    // setup remote control events in the application.
    fileprivate let remoteCommandCenter = MPRemoteCommandCenter.shared()
}


// Example providing a MediaRemote command

func enableSkipForwardCommand(interval: Int = 15) {
    remoteCommandCenter.skipForwardCommand.preferredIntervals = [NSNumber(value: interval)]
    remoteCommandCenter.skipForwardCommand.addTarget(self, action: 
        #selector(RemoteCommandManager.
            handleSkipForwardCommandEvent(event:)))
    remoteCommandCenter.skipForwardCommand.isEnabled = true
}

// Example providing a MediaRemote command

func enableSkipForwardCommand(interval: Int = 15) {
    remoteCommandCenter.skipForwardCommand.preferredIntervals = [NSNumber(value: interval)]
    remoteCommandCenter.skipForwardCommand.addTarget(self, action: #selector(RemoteCommandManager.
        handleSkipForwardCommandEvent(event:)))
    remoteCommandCenter.skipForwardCommand.isEnabled = true
}
func enableSkipForwardCommand(interval: Int = 15) {
    remoteCommandCenter.skipForwardCommand.preferredIntervals = [NSNumber(value: interval)]
    remoteCommandCenter.skipForwardCommand.addTarget(self, action: #selector(RemoteCommandManager.handleSkipForwardCommandEvent(event:)))
    remoteCommandCenter.skipForwardCommand.isEnabled = true
}
Audio Experience
Audio Experience

Auto launch

Frontmost App state

Notifications

Shortcuts
Auto Launch
Auto Launch

Auto launch Audio Apps
Auto Launch

Auto launch Audio Apps

Now Playing session on iPhone brings Apple Watch app frontmost
Auto Launch

Auto launch Audio Apps

Now Playing session on iPhone brings Apple Watch app frontmost

Stays frontmost for duration of the session
Auto Launch
Now Playing session API
Auto Launch
Now Playing session API

You will know when you’re launched for a Now Playing session on iPhone
Auto Launch
Now Playing session API

You will know when you’re launched for a Now Playing session on iPhone
Take your user directly to the view
Auto Launch
Now Playing session API

You will know when you’re launched for a Now Playing session on iPhone
Take your user directly to the view

Use `handleRemoteNowPlayingActivity()` on `WKExtensionDelegate`
class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func handleRemoteNowPlayingActivity() {
        // Get visible controller
        let visibleController = WKExtension.shared().visibleInterfaceController
        if (visibleController?.isKind(of: RemoteControlInterfaceController.self) ?? false) {
            // User is already at our NowPlaying UI. Log this condition
        } else {
            WKInterfaceController.reloadRootPageControllers(
                withNames: ["remoteControlInterfaceController"],
                contexts: nil,
                orientation: .horizontal,
                pageIndex: 0)
        }
    }
}
class ExtensionDelegate: NSObject, WKExtensionDelegate {
    func handleRemoteNowPlayingActivity() {
        // Get visible controller
        let visibleController = WKExtension.shared().visibleInterfaceController
        if (visibleController?.isKind(of: RemoteControlInterfaceController.self) ?? false {
            // User is already at our NowPlaying UI. Log this condition
        } else {
            WKInterfaceController.reloadRootPageControllers(
                withNames: ["remoteControlInterfaceController"],
                contexts: nil,
                orientation: .horizontal,
                pageIndex: 0)
        }
    }
}
class ExtensionDelegate: NSObject, WKExtensionDelegate {
    func handleRemoteNowPlayingActivity() {
        // Get visible controller
        let visibleController = WKExtension.shared().visibleInterfaceController
        if (visibleController?.isKind(of: RemoteControlInterfaceController.self) ?? false {
            // User is already at our NowPlaying UI. Log this condition
        } else {
            WKInterfaceController.reloadRootPageControllers(
                withNames: \"remoteControlInterfaceController\",
                contexts: nil,
                orientation: .horizontal,
                pageIndex: 0)
        }
    }
}
class ExtensionDelegate: NSObject, WKExtensionDelegate {
    func handleRemoteNowPlayingActivity() {
        // Get visible controller
        let visibleController = WKExtension.shared().visibleInterfaceController
        if (visibleController?.isKind(of: RemoteControlInterfaceController.self) ?? false {
            // User is already at our NowPlaying UI. Log this condition
        } else {
            WKInterfaceController.reloadRootPageControllers(
                withNames: "remoteControlInterfaceController",
                contexts: nil,
                orientation: .horizontal,
                pageIndex: 0)
        }
    }
}
Auto Launch
Opt out
Auto Launch
Opt out

App can opt out
Auto Launch
Opt out

App can opt out

Do the right thing
Auto Launch
Opt out

App can opt out

Do the right thing

Info.plist key
Auto Launch
Opt out

App can opt out

Do the right thing

Info.plist key

Now Playing app will show if you have opted out
Active means "state" - match template.
Frontmost App State
Frontmost App State

WatchConnectivity resumes
Frontmost App State

WatchConnectivity resumes
URLSession resumes
Frontmost App State

WatchConnectivity resumes

URLSession resumes

Frontmost notification
Frontmost App State

WatchConnectivity resumes

URLSession resumes

Frontmost notification

Haptics
Frontmost App State

WatchConnectivity resumes

URLSession resumes

Frontmost notification

Haptics
Audio Experience
Frontmost
Audio Experience
Frontmost

Kept frontmost while playing audio
Audio Experience

Frontmost

Kept frontmost while playing audio

User navigates away
Audio Experience

Frontmost

Kept frontmost while playing audio

User navigates away

• Background playback still occurs
Audio Experience
Frontmost

Kept frontmost while playing audio

User navigates away
• Background playback still occurs

Properly handle background events
Audio Experience
Notifications

Content available local notification
Audio Experience
Notifications

Content available local notification

Play option as the primary action
Audio Experience
Notifications

Content available local notification

Play option as the primary action

Configuration step
Audio Experience
Notifications

Content available local notification

Play option as the primary action

Configuration step
class DownloadManager : NSObject, URLSessionTaskDelegate, URLSessionDownloadDelegate {
    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didWriteData bytesWritten: Int64, totalBytesWritten: Int64, totalBytesExpectedToWrite: Int64) {
        if totalBytesExpectedToWrite > 0 {
            let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
            // Update UI showing progress
        }
    }

    func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didFinishDownloadingTo location: URL) {
        // Manage file
        // Post notification alerting user of playback opportunity
    }

    func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: Error?) {
        // Clean up state, handle errors if there are any
    }
}
class DownloadManager : NSObject, URLSessionTaskDelegate, URLSessionDownloadDelegate {
  func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didWriteData bytesWritten: Int64, totalBytesWritten: Int64, totalBytesExpectedToWrite: Int64) {
    if totalBytesExpectedToWrite > 0 {
      let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
      // Update UI showing progress
    }
  }

  func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask, didFinishDownloadingTo location: URL) {
    // Manage file
    // Post notification alerting user of playback opportunity
  }

  func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: Error?) {
    // Clean up state, handle errors if there are any
  }
}
Audio Experience
Frontmost notifications
Audio Experience
Frontmost notifications

```swift
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void) {
```
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent
notification: UNNotification, withCompletionHandler completionHandler:
@escaping (UNNotificationPresentationOptions) -> Void)

Play haptic
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void)

Play haptic
Donations

User Action $\rightarrow$ INMediaPlaybackIntent $\rightarrow$
Hey Siri
museum tour
Making a Great Experience

Glanceable Information:
- MuseumTour
  - Dalí exhibit coming soon

Tappable Actions:
- MuseumTour
  - Play Audio tour
Audio Experience

Shortcuts
Audio Experience

Shortcuts

Donate INMediaPlaybackIntent
Audio Experience

Shortcuts

Donate INMediaPlaybackIntent

Relevant shortcut API
Audio Experience

Shortcuts

Donate INMediaPlaybackIntent

Relevant shortcut API

Shortcut phrases
Audio Experience

Shortcuts

Donate INMediaPlaybackIntent

Relevant shortcut API

Shortcut phrases

<table>
<thead>
<tr>
<th>Introduction to Siri Shortcuts</th>
<th>WWDC 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building for Voice with Siri Shortcuts</td>
<td>WWDC 2018</td>
</tr>
<tr>
<td>Shortcuts on the Siri Watch Face</td>
<td>WWDC 2018</td>
</tr>
</tbody>
</table>
Summary
Summary

Native controls
Summary

Native controls

Background playback
Summary

Native controls
Background playback
Progress
More Information
