The Life of a watchOS App

Session 216

Neil Desai, WatchKit Framework Engineer
Unified process runtime
Frontmost app state
Background app refresh
Background modes
Unified process runtime
Frontmost app state
Background app refresh
Background modes
Unified process runtime
Frontmost app state
Background app refresh
Background modes
Unified process runtime
Frontmost app state
Background app refresh
Background modes
Unified process runtime
Frontmost app state
Background app refresh
Background modes
Unified Process Runtime
Architecture

Apple Watch
Architecture

Apple Watch

UI

Watch App
Unified Process Runtime

Performance win
Unified Process Runtime

Performance win

• Touch latency improvements
Unified Process Runtime

Performance win

• Touch latency improvements
• Pan performance
Unified Process Runtime

Performance win
• Touch latency improvements
• Pan performance
• Launch performance
Unified Process Runtime

Performance win
• Touch latency improvements
• Pan performance
• Launch performance

Upped-memory limits accordingly
No changes required.
Frontmost App State
Apple Pie Me
Order a Pie!
Frontmost App State
Frontmost App State

Application state
Frontmost App State

Application state

Frontmost
Frontmost App State

Application state

Frontmost

Enhanced capabilities
Frontmost App State

Application state

Frontmost

Enhanced capabilities

Architecture
Order a Pie!
// Enable Extended Frontmost

override func willActivate() {
    WKExtension.shared().isFrontmostTimeoutExtended = true
}

// Enable Extended Frontmost

override func willActivate() {
    WKExtension.shared().isFrontmostTimeoutExtended = true
}

Great, but what does this mean?
Enhanced Capabilities
Enhanced Capabilities

WatchConnectivity resumes
Enhanced Capabilities

WatchConnectivity resumes

NSURLSession resumes
Enhanced Capabilities

- WatchConnectivity resumes
- NSURLSession resumes
- Task completion
Enhanced Capabilities

WatchConnectivity resumes

NSURLSession resumes

Task completion

Haptics
Enhanced Capabilities

WatchConnectivity resumes

NSURLSession resumes

Task completion

Haptics

Frontmost notification
Benefits of Being Frontmost

WatchConnectivity background transfer improvements
Benefits of Being Frontmost

WatchConnectivity background transfer improvements

NSURLSession transfer improvements
Benefits of Being Frontmost

WatchConnectivity background transfer improvements

NSURLSession transfer improvements

• Initiation of a download occurs immediately most times
Benefits of Being Frontmost

Task completions
Benefits of Being Frontmost

Task completions

• Prioritized
Benefits of Being Frontmost
Benefits of Being Frontmost

Haptics can be used to tap your user while frontmost
Benefits of Being Frontmost

Haptics can be used to tap your user while frontmost

Speaker conditions
Benefits of Being Frontmost

Haptics can be used to tap your user while frontmost

Speaker conditions

Bluetooth Headphone conditions
Benefits of Being Frontmost
Benefits of Being Frontmost

Receive remote and local
Benefits of Being Frontmost

Receive remote and local

Choose the right experience for your user
Benefits of Being Frontmost

Receive remote and local

Choose the right experience for your user

```swift
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void)
```
All of this is free.
Architect your apps for this experience.
User Model

Order → View Order → Eat Pie!
User/App Model

Order

Receive Order

View Order

Pie’s Cooking

Eat Pie!
User/App Model

Order

Receive Order

Pie’s Cooking

View Order

Courier on the Way

Eat Pie!
User/App Model

Order
Receive Order

View Order
Pie’s Cooking

Courier on the Way

Courier Outside!
Eat Pie!
Receive Order
Apple Pie Me

10:09

We’ll tap you when your pie is ready!
We'll tap you when your pie is ready!
Timeline

Post Order
Timeline

Post Order

Enable Timeout API
Timeline

Post Order

Enable Timeout API

Update UI
Oh no! Something went wrong with your order.
Timeline

- Post Order
- Enable Timeout API
- Update UI
- Schedule Fallback Local Notification
@IBAction func orderButtonPressed() {
    let session = MyNSURLSessionDelegate.backgroundSession
    let task = session.downloadTask(with: orderPostURL)
    WKExtension.shared().isFrontmostTimeoutExtended = true
    triggerFallbackLocalNotification()
    WKInterfaceController.reloadRootPageControllers(withNames: ["orderController"],
        contexts: nil,
        orientation: .horizontal,
        pageIndex: 0)
}
@IBAction func orderButtonPressed() {

    let session = MyNSURLSessionDelegate.backgroundSession
    let task = session.downloadTask(with: orderPostURL)
    WKExtension.shared().isFrontmostTimeoutExtended = true
    triggerFallbackLocalNotification()
    WKInterfaceController.reloadRootPageControllers(withNames: [
        "orderController",
        contexts: nil,
        orientation: .horizontal,
        pageIndex: 0)
}

@IBAction func orderButtonPressed() {

    let session = MyNSURLSessionDelegate.backgroundSession
    let task = session.downloadTask(with: orderPostURL)

    WKExtension.shared().isFrontmostTimeoutExtended = true
    triggerFallbackLocalNotification()

    WKInterfaceController.reloadRootPageControllers(withNames: ["orderController"],
        contexts: nil,
        orientation: .horizontal,
        pageIndex: 0)
}

@IBAction func orderButtonPressed() {
    let session = MyNSURLSessionDelegate.backgroundSession
    let task = session.downloadTask(with: orderPostURL)
    WKExtension.shared().isFrontmostTimeoutExtended = true
    triggerFallbackLocalNotification()
    WKInterfaceController.reloadRootPageControllers(withNames: ["orderController"],
        contexts: nil,
        orientation: .horizontal,
        pageIndex: 0)
}
@IBAction func orderButtonPressed() {
    let session = MyNSURLSessionDelegate.backgroundSession
    let task = session.downloadTask(with: orderPostURL)
    WKExtension.shared().isFrontmostTimeoutExtended = true
    triggerFallbackLocalNotification()
    WKInterfaceController.reloadRootPageControllers(withNames: ["orderController"],
    contexts: nil,
    orientation: .horizontal,
    pageIndex: 0)
}
@IBAction func orderButtonPressed() {
    let session = MyNSURLSessionDelegate.backgroundSession
    let task = session.downloadTask(with: orderPostURL)
    WKExtension.shared().isFrontmostTimeoutExtended = true
    triggerFallbackLocalNotification()
    WKInterfaceController.reloadRootPageControllers(withNames: \"orderController\",
        contexts: nil,
        orientation: .horizontal,
        pageIndex: 0)
}
User Model

Receive Order
User Model

Receive Order

User Puts Wrist Down

Pie’s Cooking
Pie’s Cooking

Apple Pie Me 10:09

We’ll tap you when your pie is ready!
override func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void) {

cancelFallbackNotifications()

WKInterfaceController.reloadRootPageControllers(withNames: ["cookingController"], contexts: nil, orientation: .horizontal, pageIndex: 0)

WKInterfaceDevice.current().play(.success)

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

cancelFallbackNotifications()

WKInterfaceController.reloadRootPageControllers(withNames: ["cookingController"],
contexts: nil,
orientation: .horizontal,
pageIndex: 0)

WKInterfaceDevice.current().play(.success)

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

    cancelFallbackNotifications()
    WKInterfaceController.reloadRootPageControllers(withNames: ["cookingController"], contexts: nil, orientation: .horizontal, pageIndex: 0)

    WKInterfaceDevice.current().play(.success)
    completionHandler()
}
override func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void) {

    cancelFallbackNotifications()

    WKInterfaceController.reloadRootPageControllers(withNames: 
        ["cookingController"], contexts: nil, orientation: .horizontal, pageIndex: 0)

    WKInterfaceDevice.current().play(.success)

    completionHandler()
}


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

    cancelFallbackNotifications()
    WKInterfaceController.reloadRootPageControllers(withNames: [
        "cookingController"],
        contexts: nil,
        orientation: .horizontal,
        pageIndex: 0)

    WKInterfaceDevice.current().play(.success)
    completionHandler()
}
override func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void) {

    cancelFallbackNotifications()
    WKInterfaceController.reloadRootPageControllers(withNames: ["cookingController"], contexts: nil, orientation: .horizontal, pageIndex: 0)

    WKInterfaceDevice.current().play(.success)
    completionHandler()
}
User Model

Pie’s Cooking

Minutes
Your pie is cooking!
User Model

User Raises Wrist

Pie’s Cooking
App Model

Pie’s Cooking

ETA

Courier on the Way
User Model

Courier on the Way
User Model

User Raises Wrist

Courier on the Way
User Model

User Raises Wrist

Courier on the Way

Courier Outside!
User Model

Order

Receive Order

Pie’s Cooking

View Order

Courier on the Way

Courier Outside!

Eat Pie!
User Model

顺序：
- 订单 (Order)
- 收到订单 (Receive Order)
- 皮的烹饪 (Pie’s Cooking)
- 快递在路上 (Courier on the Way)
- 快递在外面 (Courier Outside!)
- 查看订单 (View Order)
- 返回至手表面 (Back to Watch Face)
- 吃派 (Eat Pie!)
Timeline

Receive Order
Timeline

Receive Order

Pie’s Cooking
Timeline

Receive Order

Pie’s Cooking

Courier on the Way
Review

Order
Receive Order

View Order
Pie’s Cooking

Courier on the Way

Courier Outside!
Eat Pie!
Background App Refresh
Background App Refresh

Overview

watchOS

Task

Applications

Suspended
Background App Refresh

Overview

watchOS

Task

Task

Task

Task

Task

Task

Applications

Suspended
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>)
Background App Refresh

Overview

watchOS

Applications

Suspended
API Changes
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
    for task in backgroundTasks {
        switch task {
        case let backgroundTask as WKApplicationRefreshBackgroundTask:
            // Perform functions as needed
            WKExtension.shared().scheduleSnapshotRefresh(withPreferredDate: fireDate,
                userInfo: nil) { error in
                // code
            }
            backgroundTask.setTaskCompleted()
        case ...// handle other task types
        }
    }
}
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
    for task in backgroundTasks {
        switch task {
            case let backgroundTask as WKApplicationRefreshBackgroundTask:
                // Perform functions as needed
                WKExtension.shared().scheduleSnapshotRefresh(withPreferredDate: fireDate,
                    userInfo: nil) { error in
                        // code
                    }
                backgroundTask.setTaskCompleted()
            case ... // handle other task types
        }
    }
}
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
    for task in backgroundTasks {
        switch task {
            case let backgroundTask as WKApplicationRefreshBackgroundTask:
                // Perform functions as needed
                WKExtension.shared().scheduleSnapshotRefresh(withPreferredDate: fireDate,
                    userInfo: nil) { error in
                    // code
                }
                backgroundTask.setTaskCompleted()
            case ... // handle other task types
        }
    }
}
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
    for task in backgroundTasks {
        switch task {
        case let backgroundTask as WKApplicationRefreshBackgroundTask:
            // Perform functions as needed
            backgroundTask.setTaskCompletedWithSnapshot(true)
        case ... // handle other task types
        }
    }
}
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
    for task in backgroundTasks {
        switch task {
        case let backgroundTask as WKApplicationRefreshBackgroundTask:
            // Perform functions as needed
            backgroundTask.setTaskCompletedWithSnapshot(true)
        case ... // handle other task types
        }
    }
}
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
    for task in backgroundTasks {
        switch task {
        case let snapshotTask as WKSnapshotRefreshBackgroundTask:
            // Perform functions as needed
            // Snapshot tasks have a unique completion call, make sure to set
            // your expiration date
            snapshotTask.setTaskCompleted(restoredDefaultState: true,
                                           estimatedSnapshotExpiration: Date.distantFuture,
                                           userInfo: nil)
        case ... // handle other task types
        }
    }
}
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
    for task in backgroundTasks {
        switch task {
        case let snapshotTask as WKSnapshotRefreshBackgroundTask:
            // Perform functions as needed
            // Snapshot tasks have a unique completion call, make sure to set
            // your expiration date
            snapshotTask.setTaskCompleted(restoredDefaultState: true,
                                          estimatedSnapshotExpiration: Date.distantFuture,
                                          userInfo: nil)

        case ... // handle other task types
        }
    }
}
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
    for task in backgroundTasks {
        switch task {
            case let snapshotTask as WKSnapshotRefreshBackgroundTask:
                // Perform functions as needed
                snapshotTask.setTaskCompletedWithSnapshot(true)
            case ... // handle other task types
        }
    }
}
func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>) {
    for task in backgroundTasks {
        switch task {
            case let snapshotTask as WKSnapshotRefreshBackgroundTask:
                // Perform functions as needed
                snapshotTask.setTaskCompletedWithSnapshot(true)
            case ... // handle other task types
        }
    }
}
// Deprecated in watchOS 4.0
// Not called in watchOS 3, if you implement handle(_ backgroundTasks:)

public protocol CLKComplicationDataSource: NSObjectProtocol {
    optional public func getNextRequestedUpdateDate(handler: @escaping (Date?) -> Swift.Void)
    optional public func requestedUpdateDidBegin()
    optional public func requestedUpdateBudgetExhausted()
}
// Use for Scheduling Complication Updates in watchOS 3 and Greater

public protocol WKExtensionDelegate: NSObjectProtocol {
    optional public func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>)
}

extension WKExtension {
    open func scheduleBackgroundRefresh(withPreferredDate preferredFireDate: Date,
                                         userInfo: NSSecureCoding?,
                                         scheduledCompletion: @escaping (Error?) -> Swift.Void)
}
// Use for Scheduling Complication Updates in watchOS 3 and Greater

public protocol WKExtensionDelegate: NSObjectProtocol {
    optional public func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>)
}

extension WKExtension {
    open func scheduleBackgroundRefresh(withPreferredDate preferredFireDate: Date,
                                          userInfo: NSSecureCoding?,
                                          scheduledCompletion: @escaping (Error?) -> Swift.Void)
}
public protocol WKExtensionDelegate: NSObjectProtocol {
    optional public func handle(_ backgroundTasks: Set<WKRefreshBackgroundTask>)
}

extension WKExtension {
    open func scheduleBackgroundRefresh(withPreferredDate preferredFireDate: Date, userInfo: NSSecureCoding?, scheduledCompletion: @escaping (Error?) -> Swift.Void)
}
Background App Refresh

Tips
Background App Refresh

Tips

Schedule
Background Refresh

7:15PM

8:00PM
Background App Refresh

Tips

Schedule
Background Refresh
NSURLSession

7:15PM 7:30PM 8:00PM
Background App Refresh

Tips

Schedule
Background Refresh
NSURLConnection
Update
Complication

7:15PM
7:30PM
7:45PM
8:00PM
// New NSURLSessionTask API

class URLSessionTask: NSObject, NSCopying, NSProgressReporting {
    var earliestBeginDate : Date?
}

// New NSURLSessionTask API

class URLSessionTask: NSObject, NSCopying, NSProgressReporting {
    var earliestBeginDate: Date?
}

Background App Refresh

Tips

7:15PM  8:00PM
Schedule Background Refresh

7:15PM - 8:00PM
Background App Refresh

Tips

Schedule Background Refresh
7:15PM

Update Complication
7:45PM

NSURLConnection
7:30PM
Apple Pie Me
Background App Refresh
Apple Pie Me

Order
Receive Order
Pie’s Cooking
Courier on the Way
Courier Outside!

View Order
Eat Pie!
Background App Refresh

Apple Pie Me

Pie’s Cooking
override func userNotificationCenter(_ center: UNUserNotificationCenter,
    willPresent notification: UNNotification,
    withCompletionHandler completionHandler: @escaping
    (UNNotificationPresentationOptions) -> Void) {

cancelFallbackNotifications()
    WKInterfaceController.reloadRootPageControllers(withNames: 
    "cookingController", contexts: nil, orientation: .horizontal, pageIndex: 0)

    WKInterfaceDevice.current().play(.success)
    completionHandler()
}
Background App Refresh

Apple Pie Me

User Raises Wrist

Pie’s Cooking

NSURLSession Get ETA
Background App Refresh

Apple Pie Me

User Raises Wrist

Pie’s Cooking

NSURLSession Get ETA

NSURLSessionTask Schedule ETA
Background App Refresh

Apple Pie Me

Order
Receive Order

View Order
Pie’s Cooking

Eat Pie!
Courier Outside!

Courier on the Way
Apple Pie Me
Apple Pie Me

NSURLSession resumes
Apple Pie Me

NSURLSession resumes

Haptics
Apple Pie Me

NSURLSession resumes

Haptics

Notifications
Apple Pie Me

NSURLSession resumes

Haptics

Notifications

Background app refresh
Apple Pie Me

NSURLSession resumes
Haptics
Notifications
Background app refresh
Schedule NSURLSession ETA requests
Background Modes
Background Modes
Background Modes

Workout
Background Modes

Workout

Audio recording
Background Modes

Workout

Audio recording

Navigation
Audio Recording Background Mode
Audio Recording Background Mode

Your UI
Audio Recording Background Mode

Your UI

Foreground initiated, background running
Audio Recording Background Mode

Your UI

Foreground initiated, background running

Frontmost while recording
Audio Recording Background Mode

Your UI

Foreground initiated, background running

Frontmost while recording

Ability to play haptics
Playback and Recording
Playback and Recording

Playback

• AVAudioPlayer (watchOS 3.1 SDK)
Playback and Recording

Playback
• AVAudioPlayer (watchOS 3.1 SDK)

Recording
• AVAudioInputNode (AVAudioEngine)
• AVAudioRecorder
• AVAudioSession recording permissions
Playback and Recording

Playback
• AVAudioPlayer (watchOS 3.1 SDK)

Recording
• AVAudioInputNode (AVAudioEngine)
• AVAudioRecorder
• AVAudioSession recording permissions

Formats supported
• AAC-LC, AAC-ELD, HE-AAC, HE-AACv2, MP3 (decoding only), Opus
Location Background Mode
Location Background Mode

Foreground initiated, background running
Location Background Mode

Foreground initiated, background running

Frontmost while in session
Location Background Mode
Location Background Mode

startUpdatingLocation
Location Background Mode

startUpdatingLocation

Set allowsBackgroundLocationUpdates
Apple Pie Me
Apple Pie Me

Driver app
Apple Pie Me

Driver app

Play haptics while in session
Summary
Summary

Unified process runtime
Summary

Unified process runtime

Frontmost app state
Summary

Unified process runtime
Frontmost app state
Background app refresh
Summary

Unified process runtime

Frontmost app state

Background app refresh

Background modes
Rethink Your Apps
Rethink Your Apps

Design apps for Frontmost App State
Rethink Your Apps

Design apps for Frontmost App State

Think through the enhanced capabilities
Rethink Your Apps

Design apps for Frontmost App State

Think through the enhanced capabilities

Build two second experiences
More Information

## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in watchOS</td>
<td>Hall 2</td>
<td>Wednesday 9:00AM</td>
</tr>
<tr>
<td>Planning a Great Apple Watch Experience</td>
<td>Grand Ballroom B</td>
<td>Thursday 5:10PM</td>
</tr>
<tr>
<td>What’s New in Audio</td>
<td>Grand Ballroom B</td>
<td>Tuesday 1:50PM</td>
</tr>
<tr>
<td>Keeping Your Watch App Up to Date</td>
<td></td>
<td>WWDC 2016</td>
</tr>
<tr>
<td>Architecting for Performance on watchOS 3</td>
<td></td>
<td>WWDC 2016</td>
</tr>
<tr>
<td>Labs</td>
<td>Technology Lab</td>
<td>Days and Times</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Background Updates and WatchKit Lab</td>
<td>Technology Lab H</td>
<td>Wed 3:10PM–6:00PM</td>
</tr>
<tr>
<td>WatchKit Lab</td>
<td>Technology Lab C</td>
<td>Thurs 1:00PM–4:00PM</td>
</tr>
<tr>
<td>WatchConnectivity and WatchKit Lab</td>
<td>Technology Lab B</td>
<td>Fri 9:00AM–11:00AM</td>
</tr>
</tbody>
</table>