Creating Independent Watch Apps

Neil Desai, watchOS Frameworks Engineer
Independent Watch Apps
Installation
Installation

iOS app no longer includes watch app
Installation

iOS app no longer includes watch app

Each device downloads its own app
Installation

iOS app no longer includes watch app

Each device downloads its own app

Asset and variant thinning
Installation

iOS app no longer includes watch app

Each device downloads its own app

Asset and variant thinning

Downloads are smaller
Dependent Apps
Dependent Apps

Watch app download will install iPhone app to iPhone
Dependent Apps

Watch app download will install iPhone app to iPhone

watchOS app launch is blocked until iPhone app is installed
Independent Apps
Independent Apps

Watch app is installed independently
Independent Apps

Watch app is installed independently

Can now uninstall the iPhone app and Watch app can remain
Independent Apps

Watch app is installed independently

Can now uninstall the iPhone app and Watch app can remain

watchOS app with iOS app is backwards compatible
Independent Apps

Watch app is installed independently

Can now uninstall the iPhone app and Watch app can remain

watchOS app with iOS app is backwards compatible

Watch-only app is watchOS 6 or later
Enterprise Distribution
Enterprise Distribution

Xcode support
Enterprise Distribution

Xcode support

Variants
Enterprise Distribution

Xcode support

Variants

platform-identifier key
Make Your Watch Apps Independent
Demo
Apps in Xcode
Apps in Xcode

Migrate your existing Xcode project
Apps in Xcode

Migrate your existing Xcode project

Watch-only app
Apps in Xcode

Migrate your existing Xcode project

Watch-only app

Simulator experience
Debugging
Debugging

Simulator debugging is up to 10x faster
Debugging

Simulator debugging is up to 10x faster

Device debugging is up to 2x faster
Debugging

Simulator debugging is up to 10x faster

Device debugging is up to 2x faster

Proxied through iPhone
Sign In and Sign Up
Sign In and Sign Up

Sign up
Sign In and Sign Up

Sign up

Terms & Conditions, use WKAlertAction API
Sign In and Sign Up

Sign up

Terms & Conditions, use WKAlertAction API

Sign In with Apple
Sign In and Sign Up

Sign up

Terms & Conditions, use WKAlertAction API

Sign In with Apple

Your own password sign in
Sign In with Apple
Sign In with Apple

Simple and secure
Sign In with Apple

Simple and secure

No forms
Sign In with Apple

Simple and secure
No forms
No new password
Sign In with Apple

Simple and secure
No forms
No new password
Sign In with Apple

Simple and secure

No forms

No new password
Sign In with Apple

Simple and secure
No forms
No new password
Sign In with Apple

Simple and secure

No forms

No new password

Two-factor authentication for every account
Sign In with Apple

Simple and secure

No forms

No new password

Two-factor authentication for every account

No email verification
Sign In with Apple

Simple and secure
No forms
No new password
Two-factor authentication for every account
No email verification
Sign-in across devices
Sign In with Apple
Sign In with Apple

Use AuthenticationServices.framework
Sign In with Apple

Use AuthenticationServices.framework

WKInterfaceAuthorizationAppleIDButton
Sign In with Apple

Use AuthenticationServices.framework

WKInterfaceAuthorizationAppleIDButton

Introducing Sign In with Apple  Wednesday, 9:00

What’s New in Authentication  Thursday, 11:00
Sign In with Apple

Enabling Sign in with Apple allows your users to authenticate with their Apple ID.
TextField

Embed TextField
TextField

Embed TextField

Use placeholder to instruct user what to input
Embed TextField

Use placeholder to instruct user what to input

Set `textContentType`
Password AutoFill Suggestions on iOS
Password AutoFill Suggestions on iOS

Set the correct `textContentType`
Password AutoFill Suggestions on iOS

Set the correct `textContentType`

Associated domains
Password AutoFill Suggestions on iOS

Set the correct `textContentType`

Associated domains

User selection autofills both username and password
TextField

TextField

Associated domains
TextField

TextField

Associated domains

Continuity keyboard
TextField

Associated domains

Continuity keyboard

One time code
Privacy Management
Privacy Management

Calendar, contacts, motion
Privacy Management

Calendar, contacts, motion

Location

Allow “MyGreatWatch” to access your location while you are using the app? This app wants to access your location while in use and in the background.
Privacy Management

Calendar, contacts, motion

Location

Health

Health Access
“MyGreatWatch” would like to access and update your health data.
Privacy Management

Calendar, contacts, motion

Location

Health
Privacy Management

Calendar, contacts, motion

Location

Health
Privacy Management

Calendar, contacts, motion

Location

Health
Push Notifications
Push Notifications

Watch as a push target
Push Notifications

Watch as a push target

User-visible notifications
Push Notifications

Watch as a push target
User-visible notifications
Background notifications
Notifications

APNs request header

Payload

APNs
Notifications

APNs
Notifications

APNs

[Diagram of Apple Watch with APNs request header and payload]
APNs Request Header
APNs Request Header

`apns-push-type`
APNs Request Header

apns-push-type
alert
APNs Request Header

- `apns-push-type`
- `alert`
- `background`
Push Notifications
Registration and delivery
Push Notifications
Registration and delivery

WatchKit for registration
Push Notifications
Registration and delivery

WatchKit for registration

UserNotifications
Push Notifications
Registration and delivery

WatchKit for registration

UserNotifications

Background notifications delivered on WKExtensionDelegate
Push Notifications
Registration and delivery

WatchKit for registration

UserNotifications

Background notifications delivered on WKExtensionDelegate

Notification service extension support
Apple Push Notification service is a robust and highly efficient service for propagating information to devices.
Background Modes

Background Modes specifies that the app provides specific background services and must be allowed to continue running while in the background. These keys should be used sparingly and only by apps providing the indicated services. Where alternatives for running in the background exist, those alternatives should be used instead.
// WatchKit code for registration of notifications

import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func applicationDidFinishLaunching() {
        let center = UNUserNotificationCenter.current()
        center.requestAuthorization(options: [.alert, .sound]) { (granted, error) in
            // Enable or disable features based on authorization.
            if (granted) {
                WKExtension.shared().registerForRemoteNotifications()
            } else { /* Handle no access */ }
        }
    }
}
// WatchKit code for registration of notifications

import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func applicationDidFinishLaunching() {
        let center = UNUserNotificationCenter.current()
        center.requestAuthorization(options: [.alert, .sound]) { (granted, error) in
            // Enable or disable features based on authorization.
            if (granted) {
                WKExtension.shared().registerForRemoteNotifications()
            } else { /* Handle no access */ }
        }
    }
}
// WatchKit code for registration of notifications

import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func applicationDidFinishLaunching() {
        let center = UNUserNotificationCenter.current()
        center.requestAuthorization(options: [.alert, .sound]) { (granted, error) in
            // Enable or disable features based on authorization.
            if (granted) {
                WKExtension.shared().registerForRemoteNotifications()
            } else { /* Handle no access */ }
        }
    }
}
// WatchKit code for registration of notifications

import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func didRegisterForRemoteNotifications(withDeviceToken deviceToken: Data) {
        /* Forward the token to your provider, using a custom method. */
    }

    func didFailToRegisterForRemoteNotificationsWithError(_ error: Error) {
        /* Disable remote notification features */
    }
}
// WatchKit code for registration of notifications

import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func didRegisterForRemoteNotifications(withDeviceToken deviceToken: Data) {
        /* Forward the token to your provider, using a custom method. */
    }

    func didFailToRegisterForRemoteNotificationsWithError(_ error: Error) {
        /* Disable remote notification features */
    }
}
// WatchKit code for registration of notifications

import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func didRegisterForRemoteNotifications(withDeviceToken deviceToken: Data) {
        /* Forward the token to your provider, using a custom method. */
    }

    func didFailToRegisterForRemoteNotificationsWithError(_ error: Error) {
        /* Disable remote notification features */
    }
}
import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func didRegisterForRemoteNotifications(withDeviceToken deviceToken: Data) {
        /* Forward the token to your provider, using a custom method. */
    }

    func didFailToRegisterForRemoteNotificationsWithError(_ error: Error) {
        /* Disable remote notification features */
    }
}
// WatchKit code for handling background notifications

import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func didReceiveRemoteNotification(_ userInfo: [AnyHashable : Any],
        fetchCompletionHandler: @escaping (WKBackgroundFetchResult) -> Void) {

        /* Handle background notification */
        fetchCompletionHandler(.newData)
    }
}
// WatchKit code for handling background notifications

import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func didReceiveRemoteNotification(_ userInfo: [AnyHashable : Any],
                                        fetchCompletionHandler: @escaping
                                        (WKBackgroundFetchResult) -> Void) {

        /* Handle background notification */
        fetchCompletionHandler(.newData)
    }
}
// WatchKit code for handling background notifications

import WatchKit
import UserNotifications

class ExtensionDelegate: NSObject, WKExtensionDelegate {

    func didReceiveRemoteNotification(_ userInfo: [AnyHashable : Any],
        fetchCompletionHandler: @escaping (WKBackgroundFetchResult) -> Void) {

        /* Handle background notification */
        fetchCompletionHandler(.newData)
    }
}
Details

`apns-push-type` is required
Details

`apns-push-type` is required

`apns-topic` is your WatchKit app bundle identifier
Details

`apns-push-type` is required

`apns-topic` is your WatchKit app bundle identifier

Alert Coordination for duplicate notifications when sent simultaneously
Complication Pushes on watchOS
Complication Pushes on watchOS

Update your app if complication is enabled on the active watch face
Complication Pushes on watchOS

Update your app if complication is enabled on the active watch face

PushKit now available on watchOS
Complication Pushes on watchOS

Update your app if complication is enabled on the active watch face

PushKit now available on watchOS

Registration and delivery
import PushKit

func registerForComplicationPushes() {
    let pushRegistry = PKPushRegistry(queue: .main)
    pushRegistry.delegate = self
    pushRegistry.desiredPushTypes = [.complication]
}

func pushRegistry(_ registry: PKPushRegistry,
    didUpdate pushCredentials: PKPushCredentials,
    for type: PKPushType) {
    /* Forward complication token to server */
}

func pushRegistry(_ registry: PKPushRegistry, didInvalidatePushTokenFor type: PKPushType) {
    /* Handle invalidated token */
}
import PushKit

func registerForComplicationPushes() {
    let pushRegistry = PKPushRegistry(queue: .main)
    pushRegistry.delegate = self
    pushRegistry.desiredPushTypes = [.complication]
}

func pushRegistry(_ registry: PKPushRegistry,
                  didUpdate pushCredentials: PKPushCredentials,
                  for type: PKPushType) {
    /* Forward complication token to server */
}

func pushRegistry(_ registry: PKPushRegistry, didInvalidatePushTokenFor type: PKPushType) {
    /* Handle invalidated token */
}
import PushKit

func registerForComplicationPushes() {
    let pushRegistry = PKPushRegistry(queue: .main)
    pushRegistry.delegate = self
    pushRegistry.desiredPushTypes = [.complication]
}

func pushRegistry(_ registry: PKPushRegistry, 
didUpdate pushCredentials: PKPushCredentials,
for type: PKPushType) {
    /* Forward complication token to server */
}

func pushRegistry(_ registry: PKPushRegistry, 
didInvalidatePushTokenFor type: PKPushType) {
    /* Handle invalidated token */
}
import PushKit

func registerForComplicationPushes() {
    let pushRegistry = PKPushRegistry(queue: .main)
    pushRegistry.delegate = self
    pushRegistry.desiredPushTypes = [.complication]
}

func pushRegistry(_ registry: PKPushRegistry,
          didUpdate pushCredentials: PKPushCredentials,
          for type: PKPushType) {
    /* Forward complication token to server */
}

func pushRegistry(_ registry: PKPushRegistry, didInvalidatePushTokenFor type: PKPushType) {
    /* Handle invalidated token */
}
// Handle incoming complication push

import PushKit

func pushRegistry(_ registry: PKPushRegistry,
didReceiveIncomingPushWith payload: PKPushPayload,
for type: PKPushType,
completion: @escaping () -> Void) {

    /* Handle receiving complication push
     Reload complication timeline */
}

// Handle incoming complication push

import PushKit

func pushRegistry(_ registry: PKPushRegistry,
didReceiveIncomingPushWith payload: PKPushPayload,
for type: PKPushType,
completion: @escaping () -> Void) {

    /* Handle receiving complication push
    Reload complication timeline */
}

Networking

URLSession
Networking

URLSession

CloudKit
URLSession
URLSession

Migrate all WatchConnectivity usage to URLSession
Migrate all WatchConnectivity usage to URLSession

Background sessions
WatchConnectivity
WatchConnectivity

Use WatchConnectivity for companion app specific interactions
WatchConnectivity

Use WatchConnectivity for companion app specific interactions

open var isCompanionAppInstalled: Bool { get }
CloudKit
CloudKit

CKSubscription
CloudKit

CKSubscription

CloudKit notifications
CloudKit

CKSubscription

CloudKit notifications

CloudKit Best Practices  WWDC 2016
CKSubscriptions
CKSubscriptions

Subscribe to changes
CKSubscriptions

Subscribe to changes

Pushes tell you when to update
CKSubscriptions

Subscribe to changes

Pushes tell you when to update

Retrieve only what has changed
Data ➔ Subscription ➔ APNs ➔ Data
// Background push

let notificationInfo = CKSubscription.NotificationInfo()

// Set only this property

notificationInfo.shouldSendContentAvailable = true

// CloudKit will deliver a push, listen for pushes via:

func didReceiveRemoteNotification(_ userInfo: [AnyHashable : Any],
    fetchCompletionHandler: @escaping (WKBackgroundFetchResult) -> Void) {...}
// Background push

let notificationInfo = CKSubscription.NotificationInfo()

// Set only this property

notificationInfo.shouldSendContentAvailable = true

// CloudKit will deliver a push, listen for pushes via:

func didReceiveRemoteNotification(_ userInfo: [AnyHashable : Any],
    fetchCompletionHandler: @escaping (WKBackgroundFetchResult) -> Void) {...}
// Background push

let notificationInfo = CKSubscription.NotificationInfo()

// Set only this property
notificationInfo.shouldSendContentAvailable = true

// CloudKit will deliver a push, listen for pushes via:

func didReceiveRemoteNotification(_ userInfo: [AnyHashable : Any],
    fetchCompletionHandler: @escaping (WKBackgroundFetchResult) -> Void) {...}
Summary
Summary

Freedom and independence
Summary

Freedom and independence

System and developer capabilities
Summary

Freedom and independence

System and developer capabilities

Make your apps independent
More Information

developer.apple.com/wwdc19/208

SwiftUI on watchOS  Wednesday, 2:00

watchOS Independence Lab  Wednesday, 9:00