watchOS SDK: WatchConnectivity Framework

Version for watchOS 4 and iOS 11 (Introduced 2017)

Namespacing Prefix: WC

About

ClockKit facilitates the design and functionality of watchOS Clock Face Complications, including setting up a timeline for time travel, and updating the information displayed.

Compatibility

WatchConnectivity is compatible with watchOS 2.0, and iOS 9.0. Starting with iOS 9.3, an iPhone can pair more than one Apple Watch at the same time; this requires the Apple Watch to run watchOS 2.2 or later. Please note that each Apple Watch can only be paired to one iPhone.

Design Basics

ClockKit facilitates the design and functionality of watchOS Clock Face Complications, including setting up a timeline for time travel, and updating the information displayed.

Objects

Central Handler Object

`class WCSession` : NSObject, CVarArg, Equatable, Hashable

The object that initiates communication between a WatchKit extension and its companion iOS app.

The session object must be configured and activated before you attempt to send messages or obtain information about the state of the connection. Check using `isSupported()`.

Getting the Default Session

`class func isSupported()` Returns a Boolean indicating whether iOS device can use a session object.

`class var `default`: WCSession` Returns the singleton session object for the current device.

Configuring the Session

`var delegate: WCSessionDelegate?` The delegate for the session object.

`func activate()` Activates the session asynchronously.

`var activationState: WCSessionActivationState` The current activation state of the session.

Getting Information about the Paired Device

`var isPaired: Bool` A Boolean indicating whether the current iPhone is paired to an Apple Watch.

`var iOSDeviceNeedsUnlockAfterRebootForReachability: Bool` A Boolean value indicating whether the paired iPhone must be unlocked to become reachable.

`var isWatchAppInstalled: Bool` A Boolean value indicating whether the Watch app is installed on the currently paired and active Apple Watch.

`var isComplicationEnabled: Bool` A Boolean value indicating whether the Watch app's complication is in use on the currently paired and active Apple Watch.

`var watchDirectoryURL: URL?` A directory for storing information specific to the currently paired and active Watch.

Determining the Session's Reachability

`var isReachable: Bool` A Boolean value indicating whether the counterpart app is available for live messaging.

Managing Background Updates

`func updateApplicationContext([String : Any])` Sends a dictionary of values that a paired and active device can use to synchronize its state.

`var applicationContext: [String : Any]` Most recent contextual data sent to paired & active device.

`var receivedApplicationContext:[String : Any]` A dictionary containing the last update data received from a paired and active device.

Sending Messages

`func sendMessage([String : Any], replyHandler: (([String : Any]) -> Void)?, errorHandler: ((Error) -> Void)? = nil)` Sends a message immediately to the paired and active device and optionally handles a response.

`func sendMessageData(Data, replyHandler: ((Data) -> Void)?, errorHandler: ((Error) -> Void)? = nil)` Sends a data object immediately to the paired and active device and optionally handles a response.

Updating Complication Data

`var remainingComplicationUserInfoTransfers: Int` The number of remaining times you can send complication data from the iOS app to the WatchKit extension.

`func transferCurrentComplicationUserInfo([String : Any] = [:])` Sends complication-related data from the iOS app to the WatchKit extension.

Transferring Data in the Background

`func transferUserInfo([String : Any] = [:])` Sends the specified data dictionary to the counterpart.

`var outstandingUserInfoTransfers: [WCSessionUserInfoTransfer]` An array of in-progress data transfers.

Transferring Files in the Background

`func transferFile(URL, metadata: [String : Any]?)` Sends the specified file and optional dictionary to the counterpart.

`var outstandingFileTransfers: [WCSessionFileTransfer]` An array of in-progress file transfers.

`var hasContentPending: Bool` A Boolean value that indicates whether the session has more content to deliver.

Constants

`enum WCSessionActivationState` Constants indicating the activation state of a session.

`let WCErrorDomain: String` The domain for errors associated with the Watch Connectivity framework.

`struct WCError`

`enum WCError.Code` Constants for errors returned during a session.

Delegate

`protocol WCSessionDelegate {` NSObjectProtocol

A delegate protocol that defines methods for receiving messages sent by a WCSession object.

Managing Session Activation

`func session(WCSession, activationDidCompleteWith: WCSessionActivationState, error: Error?)` required
Called when the activation of a session finishes. Required.

`func sessionDidBecomeInactive(WCSession)` required
Called when the session prepares to stop communicating with the current Apple Watch. Required.

`func sessionDidDeactivate(WCSession)` required
Called after all data from the previous session has been delivered and communication with the Apple Watch has ended. Required.

Managing State Changes

`func sessionWatchStateDidChange(WCSession)` optional
Called when a feature is enabled or disabled.

`func sessionReachabilityDidChange(WCSession)` optional
Called when the reachability of the counterpart session changes.

Receiving Context Data

`func session(WCSession, didReceiveApplicationContext: [String : Any])`
Called when the session receives context data from the counterpart.

Receiving Immediate Messages

`func session(WCSession, didReceiveMessage: [String : Any])` Called when an immediate message arrives.

`func session(WCSession, didReceiveMessage: [String : Any], replyHandler: ([String : Any]) -> Void)` Called when an immediate message arrives and requires a response.

`func session(WCSession, didReceiveMessageData: Data)` Called when an immediate data message arrives.

`func session(WCSession, didReceiveMessageData: Data, replyHandler: (Data) -> Void)` Called when an immediate data message arrives and requires a response.

Managing Data Dictionary Transfers

`func session(WCSession, didReceiveUserInfo: [String : Any] = [:])` Called when a data dictionary is received successfully.

`func session(WCSession, didFinish: WCSessionUserInfoTransfer, error: Error?)` Called when a data transfer operation finished successfully or because of an error.

Managing File Transfers

`func session(WCSession, didReceive: WCSessionFile)` Called when a file is received successfully.

`func session(WCSession, didFinish: WCSessionFileTransfer, error: Error?)` Called when a file transfer finished successfully or because of an error.

`}`

WCSession State and Error Codes

`Structure WCError {`

Instance Properties

`var errorCode: Int`

`var errorUserInfo: [String : Any]`

`var localizedDescription: String`

Type Properties

`static var deliveryFailed: WCError.Code`

`static var deviceNotPaired: WCError.Code`

`static var errorDomain: String`

`static var fileAccessDenied: WCError.Code`

`static var genericError: WCError.Code`

`static var insufficientSpace: WCError.Code`

`static var invalidParameter: WCError.Code`

`static var messageReplyFailed: WCError.Code`

`static var messageReplyTimedOut: WCError.Code`

`static var notReachable: WCError.Code`

`static var payloadTooLarge: WCError.Code`

`static var payloadUnsupportedTypes: WCError.Code`

`static var sessionInactive: WCError.Code`

`static var sessionMissingDelegate: WCError.Code`

`static var sessionNotActivated: WCError.Code`

`static var sessionNotSupported: WCError.Code`

`static var transferTimedOut: WCError.Code`

`static var watchAppNotInstalled: WCError.Code`

`}`

`Enumeration WCSessionActivationState {`

Constants indicating the activation state of a session.

`case notActivated` The session is not activated. When in this state, no communication occurs between the Watch app and iOS app. It is a programmer error to try to send data to counterpart app while in this state.

`case inactive` The session was active but is transitioning to the deactivated state. The session's delegate object may still receive data while in this state, but is programmer error to try to send data to counterpart app.

`case activated` The session is active and the Watch app and iOS app may communicate with each other freely.

`}`

`Enumeration WCError.Code {`

Constants for errors returned during a session.

Constants

`case genericError` An error indicating that some unknown problem occurred.

`case sessionNotSupported` An error indicating that the current device does not support the use of session objects.

`case sessionMissingDelegate` An error indicating that the WatchKit extension does not have a valid extension delegate to process events.

`case sessionNotActivated` An error indicating that the other device has not yet activated its session.

`case deviceNotPaired` An error indicating that the current device is not paired with a counterpart device. Communication can happen only between paired devices.

`case watchAppNotInstalled` An error indicating that the Watch app is not installed on the user's Apple Watch.

`case notReachable` An error indicating that the counterpart app is not currently reachable. This error occurs when you send an immediate message to an app that is not running.

`case invalidParameter` Error indicating that a parameter was invalid.

`case payloadTooLarge` An error indicating that the item being sent exceeds the maximum size limit. This type of error can occur for both data dictionaries and files.

`case payloadUnsupportedTypes` An error indicating that a dictionary contains non property-list types.

`case messageReplyFailed` An error indicating that the reply could not be returned.

`case messageReplyTimedOut` An error indicating that the counterpart app did not return a reply in time.

`case fileAccessDenied` An error indicating that a file could not be transferred because it was inaccessible. This error can occur when the file path is invalid or the app has insufficient privileges to access the file.

`case deliveryFailed` An error indicating that the payload could not be delivered.

`case insufficientSpace` An error indicating that there is not enough space on the receiving side to store the data being sent.

`case sessionInactive` An error indicating that the session is inactive. This error occurs when you try to send data using an inactive session.

`case transferTimedOut` An error indicating that the timeout value was reached before the transfer completed.

Foundation Objects Transferred in a WCSession

`Structure Data` Foundation Overview NSData: NSObject

CustomDebugStringConvertible, CustomReflectable, CustomStringConvertible, Equatable, Hashable, MutableCollection, RandomAccessCollection, RangeReplaceableCollection, ReferenceConvertible

A byte buffer in memory.

The Data value type allows simple byte buffers to take on the behavior of Foundation objects. You can create empty or pre-populated buffers from a variety of sources and later add or remove bytes. You can filter and sort the content, or compare against other buffers. You can manipulate subranges of bytes and iterate over some or all of them.

Data bridges to the NSData class and its mutable subclass, NSMutableData. You can use these interchangeably in code that interacts with Objective-C APIs.

Creating Empty Data

`init()` Creates an empty data buffer.

`init(capacity: Int)` Creates an empty data buffer of a specified size.

`init(count: Int)` Creates a new data buffer with the specified count of zeroed bytes.

`func resetBytes(in: Range<Data.Index>)` Sets a region of the data buffer to 0.

`Structure URL` Foundation Overview NSURL: NSObject

conforms to: CustomDebugStringConvertible, CustomPlaygroundQuickLookable, CustomStringConvertible, Equatable, ReferenceConvertible

A value that identifies the location of a resource, such as an item on a remote server or the path to a local file.

You can construct URLs and access their parts. For URLs that represent local files, you can also manipulate properties of those files directly, such as changing the file's last modification date. Finally, you can pass URLs to other APIs to retrieve the contents of those URLs. For example, you can use the URLSession classes to access the contents of remote resources, as described in URL Session Programming Guide.

URLs are the preferred way to refer to local files. Most objects that read data from or write data to a file have methods that accept a URL instead of a pathname as the file reference. For example, you can get the contents of a local file URL as String by calling func init(contentsOf:encoding) throws, or as a Data by calling func init(contentsOf:options) throws.

Creating a URL from a String

`init?(string: String)` Initializes with a string.

`init?(string: String, relativeTo: URL?)` Initializes with a string, relative to another URL.

Creating a File URL

`init(fileURLWithPath: String)` Initializes a newly created file URL referencing the local file or directory at path.

`init(fileURLWithPath: String, isDirectory: Bool)` Initializes a newly created file URL referencing the local file or directory at path.

`init(fileURLWithPath: String, isDirectory: Bool, relativeTo: URL?)` Initializes a newly created file URL referencing the local file or directory at path, relative to a base URL.

`init(fileURLWithPath: String, relativeTo: URL?)` Initializes a newly created file URL referencing the local file or directory at path, relative to a base URL.

`init(fileURLWithFileSystemRepresentation: UnsafePointer<Int8>, isDirectory: Bool, relativeTo: URL?)` Initializes a newly created URL referencing the local file or directory at the file system representation of the path.

`init(fileReferenceLiteralResourceName: String)`

Creating a URL by Resolving a Bookmark

Encoding and Decoding a URL

Working with the Data Representation of a URL

Accessing the Parts of a URL

Accessing Resource Values

Working with File URLs

Adding Path Components

Adding a Path Extension

Removing Path Components

Removing a Path Extension

Creating Bookmarks

Checking Reachability

Working with Promised Items

Working with Security Scoped Resources

Comparing URLs

Describing a URL

Using Reference Types

`class NSURL` An object representing the location of a resource that bridges to URL; use NSURL when you need reference semantics or other Foundation-specific behavior.

`typealias URL.ReferenceType` An alias for this value type's equivalent reference type.

WCSession Helper Objects

`class WCSessionFile` NSObject

CVarArg, Equatable, Hashable

Information about a file currently being transferred between an iOS app and WatchKit extension.

You do not create instances of this class directly. When you initiate a file transfer using the transferFile(_:metadata:) method of your app's WCSession object, the session creates an instance when it queues the file for transfer. You can get a list of in-progress transfers initiated by your app from the session's outstandingFileTransfers property. When a file is received by your app, the corresponding file object is delivered directly to your session's delegate.

The file object includes the URL of the file on the local system and an optional dictionary of keys and values that accompany the file.

Getting the File Information

`var fileURL: URL` The URL of the file that was received.

`var metadata: [String : Any]?` A dictionary of additional information that was sent with the file.

`class WCSessionFileTransfer` NSObject

CVarArg, Equatable, Hashable

Information about in-progress file transfers.

You do not create instances of this class yourself. When you initiate a file transfer, the system creates a new file transfer object to represent the transferred file. Use that object to get the file information or to cancel the transfer as needed.

To initiate a file transfer operation, call the transferFile(_:metadata:) method of your app's WCSession object.

Getting the File Information

`var file: WCSessionFile` The file being transferred.

Managing the File Transfer

`var isTransferring: Bool` A Boolean value indicating whether the file is still being transferred.

`func cancel()` Cancels the file transfer.

`class WCSessionUserInfoTransfer` NSObject

CVarArg, Equatable, Hashable, NSSecureCoding

Information about in-progress data transfers.

You do not create instances of this class yourself. When you begin a data transfer, the system creates a new instance of this class for you. Use the created object to monitor or cancel the transfer as needed.

o initiate a file transfer operation, call the transferUserInfo(_:) or transferCurrentComplicationUserInfo(_:) method of your app's WCSession object.

Getting the Transfer Information

`var isCurrentComplicationInfo: Bool` A Boolean indicating whether the data is related to the app's complication.

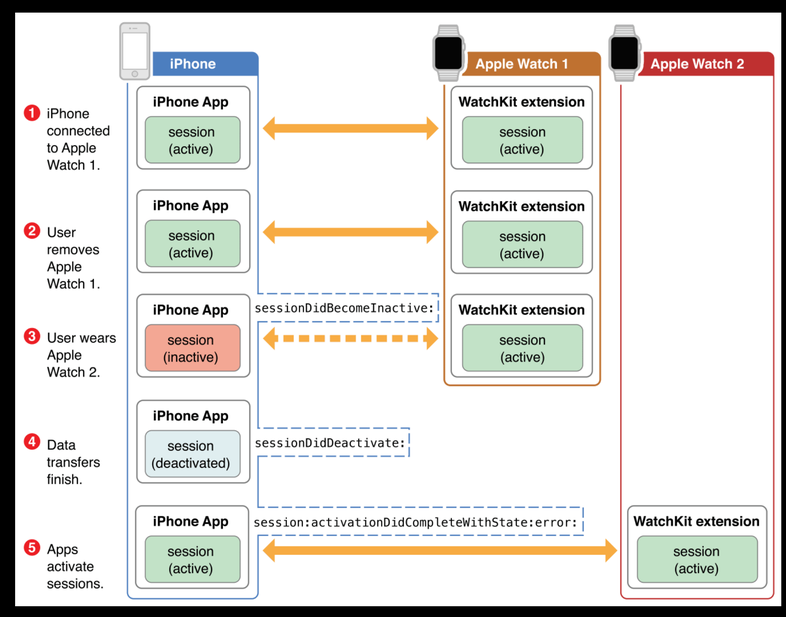
`var userInfo: [String : Any]` The data being transferred.

Managing the Transfer Operation

`var isTransferring: Bool` A Boolean value indicating whether the data is still being transferred.

`func cancel()` Cancels the data transfer.

Fundamentals



How To Set Things Up in Your Apps

Transferring real-time messages between devices

Transferring guaranteed messages between devices

Transferring application state between devices

Transferring files between devices

Updating a complication from iPhone