UNIVERSITY OF TECHNOLOGY SYDNEY

Technical report for WebKit and WatchKit

- Pascal Brokmeier - 12537015
- Piyawut Chantasrisawat - 99153025
- Tianxiang Zhan - 12096830

Content

WebKit for iOS

To watch a live demo of the developed software demo please click here for a 15 minute tutorial video.

WebKit allows developers to **display web content within their apps** without forcing the user to navigate between apps or switching to the browser. It can be considered a frameless inline version of Safari, which can be displayed at any point of the UI and renders HTML websites just like a normal browser. The framework also implements a navigation history and exposes it to the API. It offers several ways of customising the WebView, restricting a users navigation possibilities, the content that will be displayed, the local storage amount and so on.

Using this framework, applications can reuse existing web technology already developed by an organisation and therefore save developing cost. Sometimes it may also be beneficial to use HTML, e.g. when displaying documents or style sensitive graphics for which the web and CSS3 was built for. This way, a company can also reuse its existing skillset of frontend developers as well as hire professional design agencies who usually possess lots of skills in the web stack.

Technical introduction

Core objects

Starting with iOS 8, the `UIWebView` is being replaced by `WKWebView`. This is the core object the application code interacts with. Since it is a child of UIViewController it is able to be added to any existing view as a subview. The change to WKWebView is especially important, because it offers significant performance boosts for websites using WebGL, WebWorkers or other complex logic. This allows hybrid based application development using frameworks such as **cordova** or **ionic** to use more complex animations and graphics to deliver the user a native-like experience while maintaining the ability to reuse code across devices. Mainly it further closes the performance gap between native and web based applications.

The `WKBackForwardList` class manages the list of webpage previously visited by the user.

The `WKNavigation` class contains information that can be used to give the user visual feedback of the loading process of the WebView during the loading process.

Core Protocolls

WKNavigationDelegate

The `WKNavigationDelegate` implements callback methods that hook into the navigation activities of the WebView. This can be used to track changes in the location or perform in app navigations in response to a navigation change in the WebView.

The most important methods to implement are:

Callbacks that are called when the webview wants to navigate. Can programatically be allowed or cancelled using the decisionHandlers

```
- (void)webView:(WKWebView *)webView decidePolicyForNavigationAction:
(WKNavigationAction *)navigationAction decisionHandler:(void (^)
(WKNavigationActionPolicy))decisionHandler;
- (void)webView:(WKWebView *)webView decidePolicyForNavigationResponse:
(WKNavigationResponse *)navigationResponse decisionHandler:(void (^)
(WKNavigationResponsePolicy))decisionHandler;
```

Callbacks that inform the application about the navigation activities of the webView.

```
- (void)webView:(WKWebView *)webView didStartProvisionalNavigation:
(null_unspecified WKNavigation *)navigation;
- (void)webView:(WKWebView *)webView
didReceiveServerRedirectForProvisionalNavigation:(null_unspecified WKNavigation
*)navigation;
- (void)webView:(WKWebView *)webView didFailProvisionalNavigation:(null_unspecified
WKNavigation *)navigation withError:(NSError *)error;
- (void)webView:(WKWebView *)webView didCommitNavigation:(null_unspecified
WKNavigation *)navigation;
- (void)webView:(WKWebView *)webView didFinishNavigation:(null_unspecified
WKNavigation *)navigation;
- (void)webView:(WKWebView *)webView didFailNavigation:(null_unspecified
WKNavigation *)navigation withError:(NSError *)error;
```

Callback that is called in case the website requires authentication (e.g. in the form of basic authentication for REST endpoints or older home router interfaces)

```
- (void)webView:(WKWebView *)webView didReceiveAuthenticationChallenge:
(NSURLAuthenticationChallenge *)challenge completionHandler:(void (^)
(NSURLSessionAuthChallengeDisposition disposition, NSURLCredential *__nullable
credential))completionHandler;
```

WKScriptMessageHandler

The `WKScriptMessageHandler` is used to receive messages from a running website. The website can send messages to the native code using the `webkit` object on the `window` object in JavaScript.

The only required method to implement for this protocol is the `didReceiveScriptMessage`. This method is called when the JavaScript runtime calls one of the `messageHandlers` defined in the configuration of the webView. The JavaScript object is automatically translated into an Objective-C object and passed to this method.

```
- (void)userContentController:(WKUserContentController *)userContentController
didReceiveScriptMessage:(WKScriptMessage *)message;
```

Message handlers

When a `UIWebView` is instantiated, the javascript within this view has the ability to easily send messages to the backing Objective-C code. All passed JavaScript objects are autmatically serialized and converted into native Objective-C or Swift objects.

```
window.webkit.messageHandlers.{NAME}.postMessage()
```

NSAppTransportSecurity for http

In order to load all websites (not only https as allowed by default), one must add the following key to the info.plist

```
Dictionary NSAppTransportSecurity
//and this as an entry
boolean NSAllowsArbitraryLoads = YES
```

Displaying a WebView

To display a WebView, very little code is needed. Using a standard one page template in XCode, the following code can be run in the first `[UIViewControllers viewDidLoad]` method:

```
//define (empty/default) settings and size
CGRect frame = _webViewContainer.frame;
WKWebViewConfiguration *configuration = [[WKWebViewConfiguration alloc] init];
//add message handler to be exposed to javascript
[configuration.userContentController addScriptMessageHandler:self
name:@"messageQueueToNative"];
//init
_webView = [[WKWebView alloc] initWithFrame:frame configuration:configuration];
//set self as delegate for the webview
_webView.navigationDelegate = self;

//configure a request URL to load
NSString *startUrl = @"http://localhost:8100"; //for demo only. links to locally
hosted mobile webapp
//set navigation url and load
[self webViewTo:startUrl];
[_webViewContainer addSubview:_webView];

//local helper function for url textField
[self urlChangedTo:startUrl];
```

Use cases and relevance to market

The WebView element is the core requirement for hybrid web applications. Without this element, a hybrid application could not run in a native environment, since there would be no way to display web content in the native app. Frameworks such as **Cordova (PhoneGap)** use this to create an app, that wrapps a website and displays it as a full screen application. The user then doesn't know, he's looking at a web based application, unless the UX makes it obvious. Frameworks such as **ionic** aim to ensuring the app experience of a user is equivalent to the experience in a truely native application. Adapters enable the use of more advanced features such as the Gyroscope, Location access, Camera access and other APIs exposed to normal iOS applications.

Hybrid applications are an essential part of the app ecosystem. The Ionic framework was created in 2014 and this was their retrospective after 1 year:

> *Over 320,000 apps have been created this year, and some particularly impressive ones were featured by both Apple and Google in their respective app stores. Ionic is being used by freelancers, startups, and Fortune 50 companies alike, and in more countries than we can even count.*
>
> *Our forum has seen over 45,000 posts this year, reaching over 100,000 unique developers a month.*

This is **just one framework** after the first year (2014–15). They now have over 24k stars and 4700+ forks on Github.

There are also many plugins and adapters for hybrid applications to make use of native technologies from within the web environment. Some examples are:

- Beacon
- Bluetooth Low Energy
- Clipboard
- Contacts
- HealthKit for iOS
- Keychain
- Native Audio
- Preferences
- Push Notifications - V5
- SMS
- Splashscreen
- SQLite
- Touchid
- Vibration

Technically one could write an application purely based on Javascript, that uses a small adapter to access an Apple Watch using the WatchKit. Even more helpful would be a home automation app, that accesses the home's devices through HomeKit using native objective-c code. All the logic for displaying and managing the assets however would be written in web technologies. This way, the application can also run on an Android, where it would then use the Nest APIs, Weave or Brillo, the pendants to Apples HomeKit

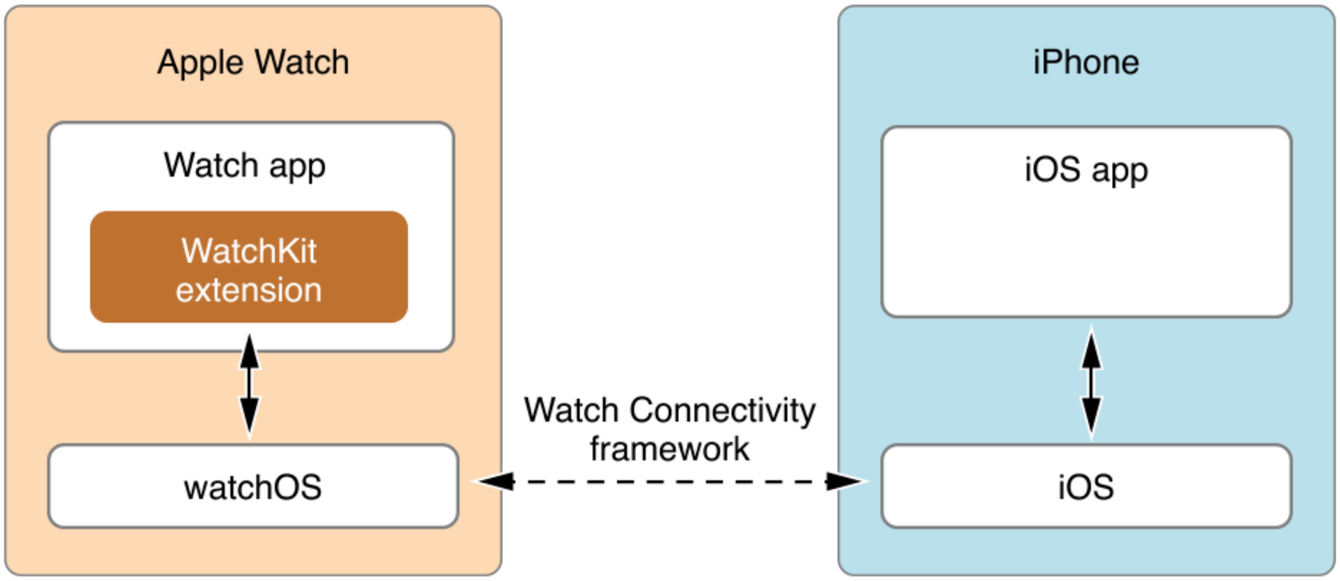
WatchKit for iOS

During Apple's WWDC 2014, Apple has released their new product, Apple Watch, along with its SDK WatchKit. WatchKit is a framework that helps developers to create their Apple Watch application on xCode using Obejctive-C or Swift.
The process for creating a Watch app is not much different from creating an iOS app. It uses storyboard to design the layout and it has controller classes that connect with UI component for custom code. It won't require a lot of works to deploy an existing iOS app to Apple Watch. But there are some new concepts about this framework are essential for developers.

Technical introduction

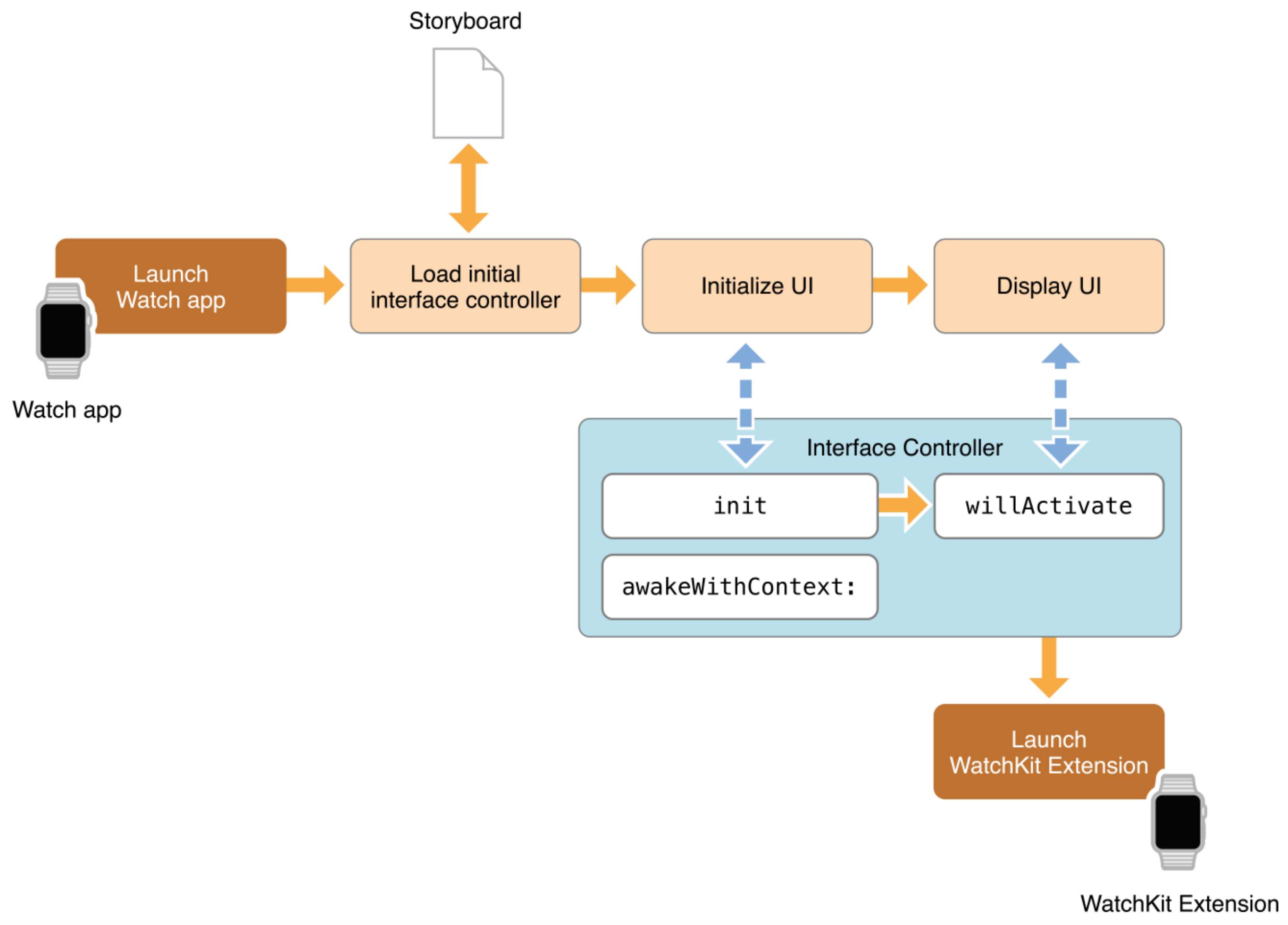
Structure of a WratchKit app

The interfaces of an application are composed of Watch application and WatchKit. When user opens an app, watchOS first loads the user interface. Meanwhile, WatchKit extension, which manages the content of interface, was launched by watchOS. The image below demonstrates the relationship between the Watch app, the WatchKit extension, and the iOS app.

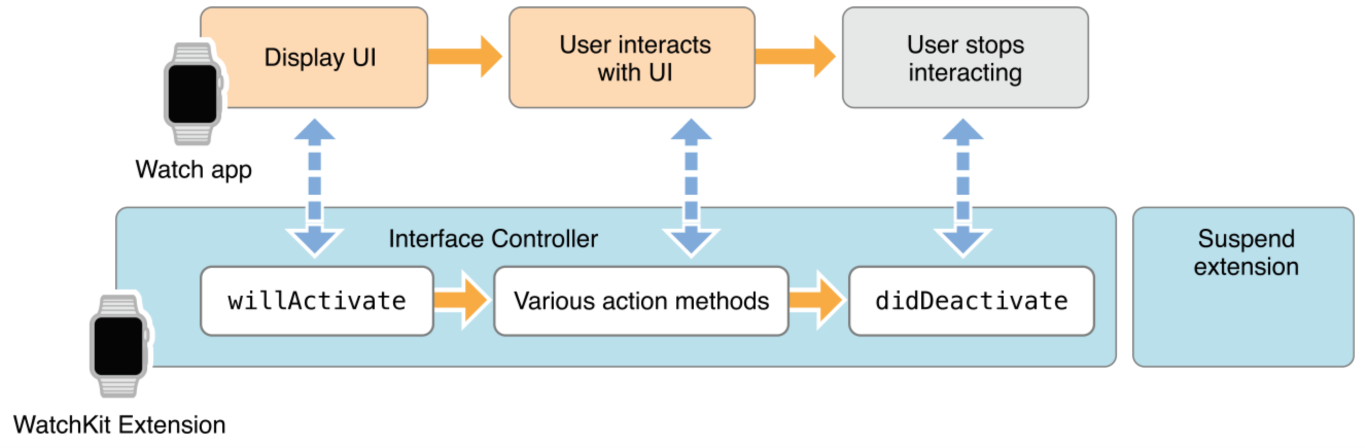


The life cycle of Watch app is drived by user interactions. There are three ways to open an app, click it from Home scren, interact with glance and coplication, or view the notifications. After opened, the app will keep alive as long as the user is interacting with it. The WatchKit extension is responsible for exchanging information. And iOS will suspend WatchKit extension if user stops interacting.

During the launch, the storyboard scene will be loaded first in response to user's interaction. Then, the WatchKit extension will create a corresponding interface controller object allows the application to display the scene.



As mentioned before, as long as the user is interactiong with the app, it keeps alive. Once the user stops interacting with Apple watch or exits the app, iOS will suspend the interface controller and the the extension.



Key classes of WatchKit

WKExtension

The behaviors between an app's interface controllers is managed by WKExtension object. When the app is launched, a WKExtension object will be created by the system. Thisobject performs app-level tasks.

WKInterfaceController

Like the UIViewController from UIKit, WKInterfaceController handles implement of Watch app's interface. The only difference is that WKInterfaceController doesn't manage any actual views. It can only manage the behavior of an interface controller remotely from inside of WatchKit extension.

WKInterfaceButton

This is the corresponding class of UIButton in WatchKit. Each object represents a button on the screen. When the button is interacted by the user, the object calls a related function.

WKInterfaceLabel

Same as UILabel, WKInterfaceLabel implements a read-only text view. Application can change the content of the label through this object. And this object can also style the text.

WCSession

In fact WCSession is not one of WatchKit classes, but this is very important in Watch application. The WCSession class performs communication between a WatchKit extension and its companion iOS app. In order to communicate, a WCSession object need to be created and activated on both sides. After activation, the two sides can send information back and forth.

Code Example

To connect Apple watch with the application on the phone, Watch Connectivity is the solution to work on it. WCSession will help to send and receive message between watch application and parent iOS application. The first thing is check that WCSessions is supported in the application or not.

First of all, we have to import WatchConnectivity and WCSessionDelegate to InterfaceController.m and input the code to activate WCSession in (void)willActivate

```
#import <WatchConnectivity/WatchConnectivity.h>

@interface InterfaceController()<WCSessionDelegate>

- (void)willActivate {
    // This method is called when watch view controller is about to be visible to
user
    [super willActivate];

    if([WCSession isSupported]){
        WCSession *session = [WCSession defaultSession];
        session.delegate = self;
        [session activateSession];
    }
}
```

is the basic way to create a WCSession. This is the step that need for every time when we want to use WCSession. Next step is to create the method to send the data by using WCSession. NSDictionary will be use in this step. Key value in NSDictionary can be changed if we want to send other data. The code to handler the error and reply from the iOS application can be write in this method by using replyHandler and errorHandler in sendMessage function.

```
- (void) sendCommand:(NSString*) cmdString{
    NSDictionary *applicationData = [[NSDictionary alloc]
initWithObjects:@[cmdString] forKeys:@[@"cmdValue"]];

    [[WCSession defaultSession] sendMessage:applicationData
                               replyHandler:^(NSDictionary *reply) {
                                   //handle reply from iPhone app here
                               }
                               errorHandler:^(NSError *error) {
                                   //catch any errors here
                               }
     ];
}
```

receive data, this method has to implemented in your code. This code will implement into ViewController.m in the parent app. The objectForKey in this code need to be the same as the key that send from NSDictionary. It will return the data in NSString and we can use it to do other job in the application. In this example, we will send the data and use it to control the WebView in the parent application.

```
- (void)session:(nonnull WCSession *)session didReceiveMessage:(nonnull
NSDictionary<NSString *,id> *)message replyHandler:(nonnull void (^)
(NSDictionary<NSString *,id> * __nonnull))replyHandler {
    NSString *cmdValue = [message objectForKey:@"cmdValue"];
    if([cmdValue isEqualToString:@"Back"]){
        [_webView goBack];
    }
    else if([cmdValue isEqualToString:@"Forward"]){
        [_webView goForward];
    }
    else if([cmdValue isEqualToString:@"Refresh"]){
        [_webView reload];
    }
    else if([cmdValue isEqualToString:@"Favourite"]){
        NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];
        NSString *favouriteURL = [prefs stringForKey:@"favouriteURL"];
        [self webViewTo:favouriteURL];
    }
    else{
    }
}
```

Practical use cases

Since Apple released the WatchKit SDK in 2014. This SDK help developer to craete the Apple Watch application to connect with the application in iPhone or iPad. Generally, developer always use Apple Watch application to send the notification from parent app to Apple Watch and use it to display some data. We can see that in the instant messaging application such as Skype, WeChat, Line or Kakao talk. Not only the instant messaging application, other apllication also use notification to help iOS users see the notifications easier by using Apple Watch and they can do some action with the watch without take their phone out.

The other thing that can make Apple Watch more interesting is using it with game. With the connectivity between Apple Watch and iOS devices. It can create some interesting game that use Apple Watch to play. Apple Watch can be a special helper to play the game. With the Apple Watch features we can create some interesting things that can be use in the game such as notify you that other player is around here and you can play with them or use your distance that you walk or run in a week to claim some special items in the game.