Juan F. Caracciolo
Jul 13, 2018 · 6 min read · Listen

IOS Deep linking: URL Scheme vs Universal Links

Everything is connected nowadays. In a world where we share links as often as we do today, your app cannot be out of the loop. Deep linking is the idea of not only having a clickable link to open up your app but a smart one that will also navigate to the desired resource. Improve your user experience implemented these useful shortcuts.

. . .

There are 2 ways of implementing deep linking in IOS: URL scheme and Universal Links. While URL schemes are a well-known way of having deep linking, Universal links are the new way Apple has implemented to easily connect your webpage and your app under the same link.

Here are comparisons for you to choose from, but I will proceed to explain both:

URL Scheme

Pros	Cons
Easy to implement out of the box	Will always ask for permission (pop up)
No extra backend required	User are not familiar with this type of URLs
	Won't work in other platforms (such as Android)
	Won't work if app is not installed

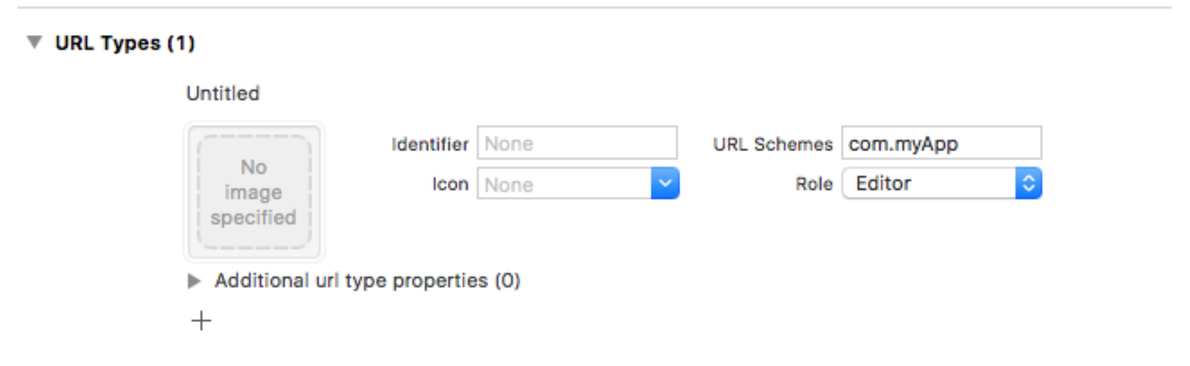
Universal Links

Pros	Cons
Won't ask for permission	Static backend with ssl needed
Won't open the browser	More complex to implement
Completely compatible with other platforms	
Fallback url if app is not installed	

Take note here, neither of these options will redirect you to the App Store if the app is not installed. **Once the App is installed**, IOS will make the connection between these URLs and the app based on the app's metadata, if the app is not installed, the metadata is here and thus the connection is never created. We will have to work that out ourselves later.

URL Schemes

Let's start with URL Schemes. URL schemes are easy to implement. All you have to do is tell the app which scheme would you like to use. For this just Open Xcode, got to *Project Settings -> Info*, and add inside '*The URL Types*" section a new URL scheme. Add something of the sort of *com.myApp* and that's it. Now install your app, open the app notes and type *com.myApp* and press enter.



Wait… nothing is happening. Well for IOS to recognize this as a link, you have to accommodate to the URL format. That is *scheme ://resource*. So we go ahead and type *com.myApp://main*. Now when we press enter, we can see IOS detected the link and when we click it, a pop up will ask for permission to open MyApp from Notes.

Now, what about if we want to redirect the user inside the app? If the user taps *com.myApp://profile* it should show the profile or *com.myApp://reset_password*, take him to the reset password screen. If we want to react once inside the app to this URL, we need to implement one method inside the **AppDelegate**

```
func application(_ app: UIApplication, open url: URL,
                 options: [UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {
    if let scheme = url.scheme,
        scheme.localizedCaseInsensitiveCompare("com.myApp") == .orderedSame,
        let view = url.host {

        var parameters: [String: String] = [:]
        URLComponents(url: url, resolvingAgainstBaseURL: false)?.queryItems?.forEach {
            parameters[$0.name] = $0.value
        }

        redirect(to: view, with: parameters)
    }
    return true
}
```

URLSchemeExample.swift hosted with ❤ by GitHub view raw

This way **com.myApp://profile?user="JuanFra"** will result in:

```
url.scheme = "com.myApp"
url.host = "profile"
parameters = [ "user" : "JuanFra" ]
```

That's all for URLSchemes. Easy right? Remember this URL **won't work if the app is not installed**, so what IOS will attempt to do is open up the link in Safari, which will obviously lead to nowhere, leaving the user looking at an empty white screen confused about what just happened. For that, we have a second option.

Universal Links

Universal links are a bit more complex. Basically, we want IOS to relate a webpage URL to our app. However, that is not so straightforward. Picture this situation. Our beloved app **"Redd1t"** is in all ways, shapes, and forms exactly like the popular social network *"Reddit"*. **Redd1t** indicates in its metadata whenever a user presses *reddit.com*, to redirect it to its app. See the problem here? IOS needs a way of validating that in fact, **Redd1t** is the owner of *www.reddit.com*, but how? Well, ask *https://www.reddit.com* of course! **Reddit.com** will need to provide a resource that indicates which is his IOS app. And it does so in the following way: IOS will make a request to https://www.reddit.com/apple-app-site-association and expect a JSON. The JSON should be as following:

```
{
    "applinks": {
        "apps": [],
        "details": [
            {
                "appID": "T5TQ36Q2SQ.com.reddit.production",
                "paths": ["*"],
            }
        ]
    }
}
```

Let's break it down for a second.

Applinks indicate this is indeed for the Universal Link declaration.

Apps should be left as an empty array (quote from Apple: *"The **apps's** key in an apple-app-site-association file must be present and its value must be an empty array*). This is most likely because these types of JSON are used for other purposes other than universal links as well, but we won't take it into consideration.

Details will then contain an array of your apps and the mapping of each subpath to the respective app.

For each app, you should add a field called **appID** which is obtained concatenating your **teamID** and your app's **bundleID**. For example, if your TeamID is *123456* and your AppID is *com.myApp*, then the result is **123456.com.myApp**.

In the **paths** field, an array of strings representing with expressions of the paths which correspond to this app. For example, if you want *myApp.com/store* to open up a different app than *myApp.com/maps*, here you can declare that * is a wildcard for any string, while **?** is a wildcard for any character. As we only have one app, any subpath will lead here, hence the *. If you want to exclude a subpath, just add **NOT** at the beginning

```
[ "/wwdc/news/", "NOT /videos/wwdc/2010/*", "/videos/wwdc/201?/*"]
```

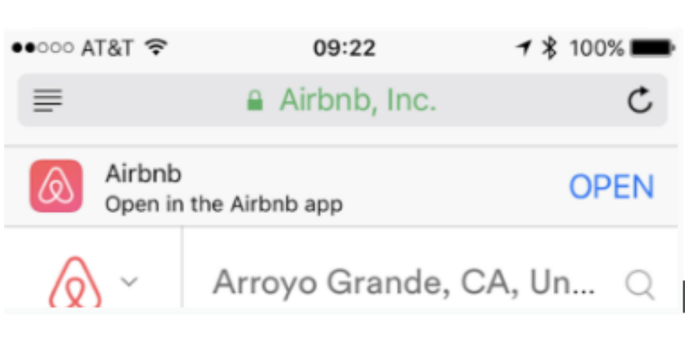
The system evaluates the path in order and will stop when finding a **NOT**, so take that into consideration when selecting the order.

In fact, if we go to https://www.reddit.com/apple-app-site-association we can see the response. Remember the link must no end in *.json*, and the request must return a header *content-type: application/json*.

If you are having trouble setting up your *apple-app-site-association*, our friends at **Branch.io** will help you out with their great validator https://branch.io/resources/aasa-validator/

Once that is done, we now must add the correct metadata to the app. First, open **Xcode**, go to *Project settings -> capabilities*. Scroll down to **Associated Domains** and turn it on. Once it is enabled, we shall add any URL that implements our *apple-app-site-association* resource, preceded by app links. Inside the Domains section, add a **applinks:myApp.com**. Once this is done, go ahead and try out your app.

If you go to My App with safari, after the app is installed, you should see a little banner indicating you that you could open up that link inside an app. If you click it the app will launch.



If you go ahead to notes and write down *www.myApp.com*, notes will recognize this as a link and let you open it. If you click it then, it will open up your app directly, not even launching safari. **This will only happen if the user taps a link, not if they visit the webpage.**

But now, say we want to redirect the user to the desired screen base on the URL parameters. We then have to implement the following method in the AppDelegate.

```
public func application(_ application: UIApplication,
                        continue userActivity: NSUserActivity,
                        restorationHandler: @escaping ([Any]?) -> Void) -> Bool {
    if let url = userActivity.webpageURL {
        var view = url.lastPathComponent
        var parameters: [String: String] = [:]
        URLComponents(url: url, resolvingAgainstBaseURL: false)?.queryItems?.forEach {
            parameters[$0.name] = $0.value
        }

        redirect(to: view, with: parameters)
    }
    return true
}
```

UniversalLinksExample.swift hosted with ❤ by GitHub view raw

What universal links allow us is not only to have a unique URL for the webpage, the IOS app and maybe even the Android App. But more than that, **Universal links allow us to have a fallback webpage if a user does not have the app installed**. If you send an email to a user, with the link *www.myApp.com/app/profile*, you could host there a static webpage that tells the user to go the App Store or even just automatically redirect it, since, **if the user has the app, the link will open it, and if not, it will fall under the webpage.**

Wrapping things up

Taking all this into consideration, **deep linking** is a crucial feature in today's apps, especially if you are sending email notifications to your users. Users consume content faster every day and the time it takes them to get to your app and navigate to the desired location is the time you are making them waste. So, what are you waiting for?