HOW CAN I GET DEEP APP SCREENS INDEXED FOR GOOGLE SEARCH?

Setting up app indexing for Android and iOS apps is pretty straightforward and well documented by Google.

Conceptually, it is a three-part process:

1. Enable your app to handle deep links.
2. Add code to your corresponding Web pages that references deep links.
3. Optimize for private indexing.

These steps can be taken out of order if the app is still in development, but the second step is crucial. Without it, your app will be set up with deep links but **not** set up for Google indexing, so the deep links will not show up in Google Search.

NOTE: iOS app indexing is still in limited release with Google, so there is a special form submission and approval process even after you have added all the technical elements to your iOS app. That being said, the technical implementations take some time; by the time your company has finished, Google may have opened up indexing to all iOS apps, and this cumbersome approval process may be a thing of the past.

Steps For Google Deep Link Indexing

Step 1: Add Code To Your App That Establishes The Deep Links

A. Pick A URL Scheme To Use When Referencing Deep Screens In Your App

App URL schemes are simply a systematic way to reference the deep linked screens within an app, much like a Web URL references a specific page on a website.

In iOS, developers are currently limited to using Custom URL Schemes, which are formatted in a way that is more natural for app design but different from Web.

In Android, you can choose from either HTTP URL schemes (which look almost exactly like web URLs) or Custom URL Schemes, or you can use both. If you have a choice and can only support one type of URL Scheme on Android, choose HTTP.

HTTP App URL Scheme For Deep Links

Android	iOS
Example App: http://example.com/hello-screen	*iOS does not currently support HTTP URL Schemes in third-party apps, so they are not used for deep linking in iOS apps (except by Apple).

Custom App URL Scheme For Deep Links

Android	iOS
Example App: example://hello-screen	Example App: example://hello-screen

B. Support That App's URL Schemes In The App

Since iOS and Android apps are built in different frameworks, different code must be added to the app to enable the deep link URL Schemes to work within the specific framework.

Android	iOS
Developers must add Intent Filters to the AndroidManifest.xml. This helps Android determine which applications are suitable for any given intent. You'll need **a separate intent filter for each deep link scheme that the app supports,** so if the app supports both HTTP and Custom URL Schemes, you will need two of each intent filter you include. An "intent" is a way to pass information between different parts of your app, or to other apps. Intent filters inform the Android Operating System which intents the app can handle receiving. Developers also need to add <action>, <data>, and <category> tags to each intent filter. These tags will not end up being part of the app deep link URL structure, but they are necessary so that the app can handle intent filters, especially from other apps. Specific requirements for each tag are available here.	Developers must declare support for the Custom URL Scheme in the app's `Info.plist` file (the iOS equivalent of the AndroidManifest.xml). They must also declare support for the `"gsd-{scheme}"` versions of the Custom URL Scheme, which are described more in Step 1D. EX: For the Example App, developers should declare support for both the Custom URL Scheme `'example'` AND `'gsd-example'` in the `Info.plist` file. 'openURL' must also be enabled in your app code (specifically in the ReceiverAppDelegate.m file) for deep links to open in the app. If you have been preparing for Apple Indexing, you may have already enabled 'openURL.' *In iOS, <action>, <data>, and <category> style controls are handled by the view controller.

C. Set Up CocoaPods

CocoaPods is a dependency management tool for iOS. It acts as a translation layer between iOS apps and the Google SDKs, so it is only necessary in iOS apps. Google has moved all its libraries to CocoaPods, and this will now be the only supported way to source them in an iOS app.

Android	iOS
*Android does not need this step, because Android apps directly communicate with Google's SDK natively already.	To enable communication between the latest version of Google's SDK and the app, developers must install CocoaPods and add the "GoogleAppIndexing" pod as a dependency in the app's Podfile.

NOTE: Developers who have never worked with CocoaPods may have to rework how they currently handle all dependent libraries in the app, because once CocoaPods is installed, it is harder and more complicated to handle other non-CocoaPods libraries. There are some iOS developers who favor CocoaPods and have been using them for some time, so your app may already be working with CocoaPods. If that's true, prepping for iOS app indexing will be much easier.

D. Enable The Back Bar

iOS devices don't come equipped with a hardware or persistent software "back" button, so Apple and Google have built workarounds to make inter-app back navigation easier. Google requires that iOS apps recognize an additional GSD Custom URL Scheme (that was set up in Step 1B). Google only uses this to trigger a "back" bar in the iOS app.

Google will generate the GSD Custom URLs automatically when someone clicks on an iOS deep link from a search result page, so we don't need to generate new GSD deep links for every screen; we just need to support the format in the `Info.plist` file and add code that will communicate with the "GoogleAppIndexing" Pod when a GSD link is received by the app.

Android	iOS

*Android phones provide a persistent “back” and “home” button in the Android OS, so this step is not necessary for Android deep linking. Inter-app backwards movement happens natively for all Android apps.	For the app to recognize the GSD links and display the back bar, you must add a line of code to the app that tells the GoogleAppIndexing pod to display a back bar when a deep link with the “GSD-“ prefix is requested.
	
	That code looks like this:
	
	`#import <GoogleAppIndexing/GoogleAppIndexing.h>`
	`...`
	
	`- (BOOL)application:(UIApplication *)application`
	`openURL:(NSURL *)URL`
	`sourceApplication:(NSString *)sourceApplication`
	`annotation:(id)annotation {`
	`NSURL *sanitizedURL = [GSDDeepLink handleDeepLink:URL];`
	`// Navigate into the appropriate view in the app using sanitizedURL`
	`...`
	
	`return YES;`

NOTE: Google’s solution is similar to Apple’s iOS 9 “Back to Search” buttons that display in the upper left portion of the phone’s Status Bar, but when it is triggered, it appears as a blue “Back Bar” that hovers over the entire phone Status Bar. The Back Bar will disappear after a short period of time if the user does not tap on it. This “disappearing” behavior also represents a unique experience for iOS deep linking in Google, since after a certain period of time, there won’t be a way for iOS users to get back to the Google Search results without switching apps manually, by clicking through the home screen. Developers compensate by adopting more tactics that pull users deeper into the app, eat up time, and distract the user from going back to Google Search until the bar disappears.

E. Set Up Robots & Google Play/Google Search Consoles

In some cases, it may make sense to generate deep links for an app screen but prevent it from showing up in search results. In Android, Google allows us to provide instructions about which screens we would like indexed for search and which we would not, but no similar mechanism is available for iOS.

Digital marketers and SEOs should use the Google Play Console and the Google Search Console to help connect your app to your website and manage app indexation. Also, double check that your website’s robots.txt file allows access to Googlebot, since it will be looking for the Web aspect of the deep links in its normal crawls.

Android	iOS
Create the `noindex.xml` file to specify which app screens you want to exclude from the index, either with URLs grouped by prefix or using a list of individual app URLs. Place it in the app's XML resources directory (`res/xml/noindex.xml`), and let Google know where you’ve put the file by referencing it in the application section of the AndroidManifest.xml. Google suggests marketers think of this like a robots noindex meta tag, but we think it behaves a bit more like X-Robots instructions in terms of its definitiveness. In the documentation, Google	**NOTE:** Google has not been super clear about how to prevent specific app screens from being indexed, probably because of the limited nature of iOS app indexing at this time. Our best recommendation at this point is to eliminate deep linking code from corresponding Web pages (i.e., skip Step 2) or block those corresponding pages in Robots.txt, so that the deep linking code or the pages themselves are not crawlable by Googlebot. This solution is only viable while Google’s app indexing is limited to apps with Web parity. Hopefully, there will be a better solution soon. We expect it to take longer for Google to begin indexing iOS apps without Web parity than Android apps without Web parity, so this may remove some of the pressure. Regardless, we are looking forward to more clarity and documentation from Google on this point soon.

emphasizes that the same `noindex.xml` file must be included with all versions of the app in the Google Play store, so **be consistent**. This noindex file is likely the one-stop-shop, no-index law of the land for your Android app.

Step 2. Add Code To Your Website That References The URL Schemes You Set Up In The App

A. Format & Validate Web Deep Links For The Appropriate App Store

Google's current app indexing process relies on Googlebot to discover and index deep links from a website crawl. Code must be added to each Web page that references a corresponding app screen.

When marking up your website, a special deep link format must be used to encode the app screen URL, along with all of the other information Google needs to open a deep link in your app. The required formatting varies slightly for Android and iOS apps and is slightly different from the URL Schemes used in the app code, but they do have some elements in common.

The `{scheme}` part of the link always refers to the URL scheme set up in your app in Step 1, and the `{host_path}` is the part of the deep link that identifies the specific app screen being referenced, like the tail of a URL. Other elements vary, as shown below:

Deep Link Formatting For Inclusion On Web

Android

Android deep links need a specific "android-app" link format that looks like this:

```
android-app://{package_name}/{scheme}/{host_path}
```

The `{package_name}` refers to the app's Application ID, which is specified in the Google Play Store. Generally, these follow a `com.[app name].android` format, which makes them easy to recognize.

Example App: http://example.com/hello-screen

```
android-app://com.GooglePlayAppID.android/http/example.com/hello-screen
```

Example App: example://hello-screen

```
android-app://com.GooglePlayAppID.android/example/hello-screen
```

iOS

iOS deep links need a similar "ios-app" special link format that looks like this:

```
ios-app://{itunes_id}/{scheme}/{host_path}
```

Use the app's iTunes ID instead of the Google Play Package Name. The iTunes ID for the app is a long string of numbers that can be obtained from iTunes Connect (the developer side of iTunes).

Example App: example://hello-screen

```
ios-app://123456/example/hello-screen
```

Deep Link Testing Tools

Android	iOS
https://developers.google.com/app-indexing/android/test	https://developers.google.com/app-indexing/ios/test

B. Add Web Deep Links To Web Pages With Corresponding App Screens

Internal app screens can be indexed when Googlebot finds deep app links in any of the following locations on your website:

- In a rel="alternate" in the HTML <head>
- In a rel="alternate" in the XML sitemap
- In Schema.org ViewAction markup

Sample code formatting for each of those indexing options is included below:

Rel=Alternate Sample Code

Android

```
<html>
<head>
  ...
  <link rel="alternate" href=" android-app://com.example.android/http/example.com/hello" />
  ...
</head>
<body> … </body>
```

*HTTP URL Scheme is included in the example code, but Custom URL Schemes are also supported.

iOS

```
<html>
<head>
  ...
  <link rel="alternate" href=" Ios-app://123456/example/hello" />
  ...
</head>
<body> … </body>
```

*Apple does not currently support HTTP URL Schemes in their apps, so this option is not included in the Rel=Alternate code sample.

XML Sitemap Sample Code

Android

```
<?xml version="1.0" encoding="UTF-8" ?>
<URLset
xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"

xmlns:xhtml="http://www.w3.org/1999/xhtml">
<URL>
  <loc>http://example.com/hello</loc>
  <xhtml:link rel="alternate" href="android-app://com.example.android/http/example.com/hello" />
...
</URLset>
```

*HTTP URL Scheme is included in the example code, but Custom URL Schemes are also supported.

iOS

```
<?xml version="1.0" encoding="UTF-8" ?>
<URLset
xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
 xmlns:xhtml="http://www.w3.org/1999/xhtml">
<URL>
  <loc>http://example.com/hello</loc>
  <xhtml:link rel="alternate" href=" Ios-app://123456/example/hello" /></URL>
...
</URLset>
```

*Apple does not currently support HTTP URL Schemes in their apps, so this option is not included in the XML Sitemap code sample.

Schema Sample Code

Android

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "WebPage",
  "@id": "http://example.com/hello",
  "potentialAction": {
    "@type": "ViewAction",
    "target": " android-app://com.example.android/http/example.com/hello"
  }
}
</script>
```

*HTTP URL Scheme is included in the example code, but Custom URL Schemes are also supported.

iOS

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "WebPage",
  "@id": "http://example.com/hello",
  "potentialAction": {
    "@type": "ViewAction",
    "target": " ios-app://123456/example/hello"
  }
}
</script>
```

*Apple does not currently support HTTP URL Schemes in their apps, so this option is not included in the Schema markup code sample.

Step 3: Optimize For Private Indexing

Both Google and Apple have a "Private" indexing feature that allows individual user behaviors to be associated with specific screens in an app. App activity that is specific to one user can be indexed on that users' phone, for private consumption only (e.g., a WhatsApp message you've viewed or an email you've opened in Mailbox).

Activities that are Privately indexed do not generate deep links that can surface in a public Google search result, but instead generate deep links that surface in other search contexts. For Android apps, this is in Chrome's autocomplete and Google Now; for iOS, this is in Spotlight, Siri, or Safari's Spotlight Suggest results.

Android

Android apps must be submitted to Google's App Indexing API and use "Activity" markup to describe content that can be surfaced in a private user's index. Google requires that Android "Activities" have a deep link URL marked up in the app to be included.

When this is in place, any deep app screen that has previously been viewed by the user can surface in Google's autocomplete feature and in Google Now Activity Cards (and soon, in Now on Tap).

iOS

Apple is placing a significant emphasis on their Private Search Indexing, which can surface app screens in Spotlight, Siri, or Safari's Spotlight Suggest -- but not in Google search results.

As described in my article on Apple Search and iOS App Indexing, Private Indexing is driven by existing app code called NSUserActivities, which must be identified as public or private. Apple's NSUserActivities associate specific user behavior with corresponding screens in the app.

URL mark-up is not required (but may be included) for the NSUserActivities to be indexed for Apple Search.

NOTE: Google's documentation seems to indicate that Activities are only used for private indexing, but Google may also use them as a measurement of engagement for more global evaluations of an app (as Apple does with NSUserActivities in Apple Search). Google has not highlighted their private indexing feature as vocally as Apple, and a user's private index can be accessed from the Phone icon in the bottom navigation of the Google Now app on Android and iOS. Currently, only Google's apps (like Gmail) are able to surface privately indexed content in organic Google search results, but we suspect this will be opened up to third-party apps in the future.