Android Developers

Handling App Links

This lesson teaches you to

- Understand URI Request Handling
- Create an Intent Handler for URIs
- Request App Links Verification
- Declare Website Associations
- Test App Links

See also

- Supporting URLs and App Indexing in Android Studio
- Creating a Statement List

Users following web links on devices are frequently presented with confusing choices. Tapping a link often results in the system asking the user which app should handle that link. For example, clicking a URI in an email from a bank might result in a dialog asking the user whether to use the browser, or the bank's own app, to open the link. Android 6.0 (API level 23) and higher allow an app to designate itself as the default handler of a given type of link. If the user doesn't want the app to be the default handler, they can override this behavior from **Settings**.

Automatic handling of links requires the cooperation of app developers and website owners. A developer must configure their app to declare associations with one or more websites, and to request that the system verify those associations. A website owner must, in turn, provide that verification by publishing (https://developers.google.com/digital-asset-links/v1/getting-started) a *Digital Asset Links* file. A Digital Asset Links file is a collection of statements conforming to the Asset Links protocol (https://github.com/google/digitalassetlinks/blob/master/well-known/details.md) that make public, verifiable assertions about other apps or websites.

The general steps for creating verified app links are as follows:

1. In your app manifest, create intent filters for your website URIs.
2. Configure your app to request verification of app links.
3. Publish a Digital Asset Links JSON file on your websites to provide verification.

Understand URI Request Handling

The app links feature allows your app to become the default handler for the website URIs you specify, as long as the user has not already chosen a default app to handle that URI pattern. When a clicked link or programmatic request invokes a web URI intent, the Android system uses the following criteria, in descending order, to determine how to handle the request:

1. **The user has set app link associations**: If the user has designated an app to handle app links, the system passes the web URI request to that app. A user can set this association in one of two ways: clicking **Always** when selecting an app from an app-selection dialog; or, opening *Settings > Apps > (gear icon) > App links*, selecting an app to use, and setting the app's **App links** property to the **Open in this app** option.
2. **The user has set no association, and there is one supporting app**: If the user has not set a preference that matches the web URI request, and there is only one app declaring support for the intent's URI pattern, the system automatically passes the request to that app.
3. **The user has set no association, and there are multiple supporting apps**: If there are multiple apps declaring support for the web URI pattern, the system displays an app-selection dialog, prompting the user to select the most appropriate app.

In case 2, if the user has newly installed the app, and the system has verified it as a handler for this type of link, the system sets the app as the default handler. In the other two cases, the presence of a verified app link handler has no effect on system behavior.

Create an Intent Handler for URIs

App links are based on the Intent (https://developer.android.com/guide/components/intents-filters.html) framework, which enables apps to handle requests from the system or other apps. Multiple apps may declare the same web link URI patterns in their intent filters. When a user clicks on a web link that does not have a default launch handler, the platform selects an app to handle the request, using the criteria described in Understanding URI Request Handling (#url-handling).

To enable your app to handle links, use intent filters in your app manifest to declare the URI patterns that your app handles. The following example shows an intent filter that can handle links to `http://www.android.com` and `https://www.android.com`:

```
<activity ...>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
```

```
        <data android:scheme="http" />
        <data android:scheme="https" />
        <data android:host="www.android.com" />
    </intent-filter>
</activity>
```

As this example shows, intent filters for app links must declare an `android:scheme` value of `http`, `https`, or both. The filter must not declare any other schemes. The filter must also include the `android.intent.action.VIEW` and `android.intent.category.BROWSABLE` category names.

This manifest declaration defines the connection between your app and a website. However, in order to have the system treat your app as the default handler for a set of URIs, you must also request that the system verify this connection. The next section explains how to implement this verification.

Request App Links Verification

In addition to using intent filters to declare an association between your app and a website, your manifest must also include an additional declaration for requesting automatic verification. When this declaration is present, the Android system attempts to verify your app after installation. If the verification succeeds, and the user has not set an alternate preference for handling your website URIs, the system automatically routes those URI requests to your app.

The system performs app-link verifications by comparing the host names in the data elements of the app's intent filters against the Digital Asset Links files (`assetlinks.json`) hosted on the respective web domains. To enable the system to verify a host, make sure that your intent filter declarations include the `android.intent.action.VIEW` intent action and `android.intent.category.BROWSABLE` intent category.

Enabling automatic verification

To enable link handling verification for your app, set the `android:autoVerify` attribute to `true` on at least one of the web URI intent filters in your app manifest, as shown in the following manifest code snippet:

```
<activity ...>

    <intent-filter android:autoVerify="true">
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="http" android:host="www.android.com" />
        <data android:scheme="https" android:host="www.android.com" />
    </intent-filter>

</activity>
```

When the `android:autoVerify` attribute is present, installing your app causes the system to attempt to verify all hosts associated with the web URIs in all of your app's intent filters. The system treats your app as the default handler for the specified URI pattern only if it successfully verifies *all* app link patterns declared in your manifest.

Supporting app linking for multiple hosts

The system must be able to verify every host specified in the app's web URI intent filters' data elements against the Digital Asset Links files hosted on the respective web domains. If any verification fails, the app is not verified to be a default handler for any of the web URI patterns defined in the app's intent filters. For example, an app with the following intent filters would fail verification if an `assetlinks.json` file were not found at both `https://www.example.com/.well-known/assetlinks.json` and `https://www.example.net/.well-known/assetlinks.json`:

```
<application>

  <activity android:name="MainActivity">
    <intent-filter android:autoVerify="true">
      <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <category android:name="android.intent.category.BROWSABLE" />
      <data android:scheme="http" android:host="www.example.com" />
      <data android:scheme="https" android:host="www.example.com" />
    </intent-filter>
  </activity>
  <activity android:name="SecondActivity">
    <intent-filter>
      <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <category android:name="android.intent.category.BROWSABLE" />
      <data android:scheme="https" android:host="www.example.net" />
    </intent-filter>
  </activity>

</application
```

Supporting app linking for multiple subdomains

The Digital Asset Links protocol treats subdomains as unique, separate hosts. If your intent filter lists both the `www.example.com` and `mobile.example.com` subdomains as hosts, you must host a separate `assetlink.json` file on each subdomain. For example, an app with the following intent filter declaration would pass verification only if the website owner published valid `assetlinks.json` files at both `https://www.example.com/.well-known/assetlinks.json` and `https://mobile.example.com/.well-known/assetlinks.json`:

```
<application>
  <activity android:name="MainActivity">
    <intent-filter android:autoVerify="true">
      <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <category android:name="android.intent.category.BROWSABLE" />
      <data android:scheme="http" android:host="www.example.com" />
      <data android:scheme="https" android:host="mobile.example.com" />
    </intent-filter>
```

```
  </activity>
</application>
```

Declare Website Associations

For app link verification to be successful, website owners must declare associations with apps. A site owner declares the relationship to an app by hosting a Digital Asset Links JSON file served with content-type `application/json` and with the name `assetlinks.json`. The owner places this file at the following well-known location on the domain:

```
https://domain[:optional_port]/.well-known/assetlinks.json
```

> **Important:** The system verifies the JSON file via the encrypted HTTPS protocol. Make sure that your hosted file is accessible over an HTTPS connection, regardless of whether your app's intent filter includes `https`.

A Digital Asset Links JSON file indicates the Android apps that are associated with the website. The JSON file uses the following fields to identify associated apps:

- `package_name`: The package name declared in the app's manifest.
- `sha256_cert_fingerprints`: The SHA256 fingerprints of your app's signing certificate. You can use the following command to generate the fingerprint via the Java keytool:

  ```
  $ keytool -list -v -keystore my-release-key.keystore
  ```

  This field supports multiple fingerprints, which can be used to support different versions of your app, such as debug and production builds.

The following example `assetlinks.json` file grants link-opening rights to a `com.example` Android app:

```
[{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
    "namespace": "android_app",
    "package_name": "com.example",
    "sha256_cert_fingerprints":
    ["14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:A0:83:42:E6:1D:BE:A8:8A:04:96:B2:
  }
}]
```

Associating a website with multiple apps

A website can declare associations with multiple apps within the same `assetlinks.json` file. The following file listing shows an example of a statement file that declares association with two apps, separately, and resides at `https://www.example.com/.well-known/assetlinks.json`:

```
[{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
    "namespace": "android_app",
    "package_name": "example.com.puppies.app",
    "sha256_cert_fingerprints":
    ["14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:A0:83:42:E6:1D:BE:A8:8A:04:96:B2:
  }
  },
  {
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
    "namespace": "android_app",
    "package_name": "example.com.monkeys.app",
    "sha256_cert_fingerprints":
    ["14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:A0:83:42:E6:1D:BE:A8:8A:04:96:B2:
  }
}]
```

Different apps may handle links for different resources under the same web host. For example, app1 may declare an intent filter for `https://example.com/articles`, and app2 may declare an intent filter for `https://example.com/videos`.

> **Note:** Multiple apps associated with a domain may be signed with the same or different certificates.

Associating multiple websites with a single app

Multiple websites can declare associations with the same app in their respective `assetlinks.json` files. The following file listings show an example of how to declare the association of example.com and example.net with app1. The first listing shows the association of example.com with app1:

```
https://www.example.com/.well-known/assetlinks.json

[{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
    "namespace": "android_app",
    "package_name": "com.mycompany.app1",
    "sha256_cert_fingerprints":
    ["14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:A0:83:42:E6:1D:BE:A8:8A:04:96:B2:
  }
}]
```

The next listing shows the association of example.net with app1. Only the very first line, which specifies the URL, is different:

```
https://www.example.net/.well-known/assetlinks.json

[{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
```

```
    "namespace": "android_app",
    "package_name": "com.mycompany.app1",
    "sha256_cert_fingerprints":
    ["14:6D:E9:83:C5:73:06:50:D8:EE:B9:95:2F:34:FC:64:16:A0:83:42:E6:1D:BE:A8:8A:04:96:B2:
  }
}]
```

Test App Links

When implementing the app linking feature, you should test the linking functionality to make sure the system can associate your app with your websites, and handle URI requests, as you expect.

To test an existing statement file, you can use the Statement List Generator and Tester (https://developers.google.com/digital-asset-links/tools/generator) tool.

Confirm the list of hosts to verify

When testing, you should confirm the list of associated hosts that the system should verify for your app. Make a list of all web URIs whose corresponding intent filters include the following attributes and elements:

- `android:scheme` attribute with a value of `http` or `https`
- `android:host` attribute with a domain URI pattern
- `android.intent.action.VIEW` category element
- `android.intent.category.BROWSABLE` category element

Use this list to check that a Digital Asset Links JSON file is provided on each named host and subdomain.

Confirm the Digital Asset Links files

For each website, use the Digital Asset Links API to confirm that the Digital Asset Links JSON file is properly hosted and defined:

```
https://digitalassetlinks.googleapis.com/v1/statements:list?
   source.web.site=https://<domain1>:<port>&
   relation=delegate_permission/common.handle_all_urls
```

Testing a web URI intent

Once you have confirmed the list of websites to associate with your app, and you have confirmed that the hosted JSON file is valid, install the app on your device. Wait at least 20 seconds for the asynchronous verification process to complete. Use the following command to check whether the system verified your app and set the correct link handling policies:

```
adb shell am start -a android.intent.action.VIEW \
    -c android.intent.category.BROWSABLE \
    -d "http://<domain1>:<port>"
```

Check link policies

As part of your testing process, you can check the current system settings for link handling. Use the following command to get a listing of existing link-handling policies for all applications:

```
adb shell dumpsys package domain-preferred-apps
 --or--
adb shell dumpsys package d
```

> **Note:** Make sure you wait at least 20 seconds after installation of your app to allow for the system to complete the verification process.

The command returns a listing of each user or profile defined on the device, preceded by a header in the following format:

```
App linkages for user 0:
```

Following this header, the output uses the following format to list the link-handling settings for that user:

```
Package: com.android.vending
Domains: play.google.com market.android.com
Status: always : 200000002
```

This listing indicates which apps are associated with which domains for that user:

- `Package` - Identifies an app by its package name, as declared in its manifest.
- `Domains` - Shows the full list of hosts whose web links this app handles, using blank spaces as delimiters.
- `Status` - Shows the current link-handling setting for this app. An app that has passed verification, and whose manifest contains `android:autoVerify="true"`, shows a status of `always`. The hexadecimal number after this status is related to the Android system's record of the user's app linkage preferences. This value does not indicate whether verification succeeded.

> **Note:** If a user changes the app link settings for an app before verification is complete, you may see a false positive for a successful verification, even though verification has failed. This verification failure, however, does not matter if the user explicitly enabled the app to open supported links without asking. This is because user preferences take precedence over programmatic verification (or lack of it). As a result, the link goes directly to your app, without showing a dialog, just as if verification had succeeded.

Test example

For app link verification to succeed, the system must be able to verify your app with all of the websites that you specify in your app's intent filters, and that meet the criteria for app links. The following example shows a manifest configuration with several app links defined:

```
<application>

    <activity android:name="MainActivity">
        <intent-filter android:autoVerify="true">
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
            <category android:name="android.intent.category.BROWSABLE" />
            <data android:scheme="http" android:host="www.example.com" />
            <data android:scheme="https" android:host="mobile.example.com" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.BROWSABLE" />
            <data android:scheme="http" android:host="www.example2.com" />
        </intent-filter>
    </activity>

    <activity android:name="SecondActivity">
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
            <category android:name="android.intent.category.BROWSABLE" />
            <data android:scheme="http" android:host="account.example.com" />
        </intent-filter>
    </activity>

      <activity android:name="ThirdActivity">
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:scheme="http" android:host="map.example.com" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.BROWSABLE" />
            <data android:scheme="market" android:host="example.com" />
        </intent-filter>
      </activity>

</application>
```

The list of hosts that the platform would attempt to verify from the above manifest is:

```
www.example.com
mobile.example.com
www.example2.com
account.example.com
```

The list of hosts that the platform would not attempt to verify from the above manifest is:

```
map.example.com (it does not have android.intent.category.BROWSABLE)
market://example.com (it does not have either an “http” or “https” scheme)
```