Using Trusted Web Activity

Last updated: June 1st, 2018

Trusted Web Activities are a new way to integrate *your* web-app content such as *your* PWA with *your* Android app using a protocol based on Custom Tabs.

Note: Trusted Web Activities are currently only available in Chrome Dev on Android, and should be available in stable in early Q4 2018 with Chrome 69.

Looking for the code?

- TrustedWebUtils Android Support Library API reference
- Sample using TrustedWebUtils

There are a few things that make Trusted Web Activities different from other ways to integrate web content with your app:

1. Content in a Trusted Web activity is **trusted** – the app and the site it opens are expected to come from the same developer. (This is verified using Digital Asset Links.)
2. Trusted Web activities come from the **web**: they're rendered by the user's browser, in exactly the same way as a user would see it in their browser except they are run fullscreen. Web content should be accessible and useful in the browser first.
3. Browsers are also updated independent of Android and your app – Chrome, for example, is available back to Android Jelly Bean. That saves on APK size and ensures you can use a modern web runtime. (Note that since Lollipop, WebView has also been updated independent of Android, but there are a significant number of pre-Lollipop Android users.)
4. The host app doesn't have direct access to web content in a Trusted Web activity or any other kind of web state, like cookies and `localStorage`. Nevertheless, you can

coordinate with the web content by passing data to and from the page in URLs (e.g. through query parameters, custom HTTP headers, and intent URIs.)

5. Transitions between web and native content are between **activities**. Each activity (i.e. screen) of your app is either completely provided by the web, or by an Android activity

To make it easier to test, there are currently no qualifications for content opened in the preview of Trusted Web activities. You can expect, however, that Trusted Web activities will need to meet the same Add to Home Screen requirements. You can audit your site for these requirements using the Lighthouse "user can be prompted to Add to Home screen" audit.

Today, if the user's version of Chrome doesn't support Trusted Web activities, we'll fall back to a simple toolbar like the one you'd see in a Custom Tab. It is also possible for other browsers to implement the same protocol that Trusted Web activities use. While the host app has the final say on what browser gets opened, we recommend the same policy as for Custom Tabs: use the user's default browser, so long as that browser provides the required capabilities.

We hope that you can experiment with this API and give us feedback at @ChromiumDev

Getting started

You can quickly build an app that launches a Trusted Web activity in three simple steps.

1. Set up Digital Asset Links in an Android app
2. Deploy your **`assetlinks.json`** to prove ownership of the domain
3. Create an activity that launches the Trusted Web Activity (you can use the Chrome Custom Tabs support library v27 or above to simplify the process)

This sample allows you to open an SVG in the SVGOMG PWA.

Set up Digital Asset Links in an Android app

The basis of Trusted Web activities is the developer-defined trust relationship between the Android app and the web content it launches established via Digital Asset Links. This assures the browser that the app opening a site is controlled by the same developer. In Chrome, attempting to launch a URL that you can't prove ownership over will fall back to opening a Custom Tab.

By setting **`autoVerify="true"`** in any intent filters you add to your app, links will also open by default on devices higher than API level 23.

First, add a statement to your AndroidManifest.xml:

```
<manifest>
  <application>
    ...
    <meta-data android:name="asset_statements"
android:resource="@string/asset_statements" />
    ...
  </application>
</manifest>;
```

And in res/values/strings.xml you'll need to update the statement and the configuration to point to your website:

```
<resources>
    <string name="app_name">SVGOMG</string>
    <string name="asset_statements">
        [{
            \"relation\": [\"delegate_permission/common.handle_all_urls\"],
            \"target\": {
                \"namespace\": \"web\",
                \"site\": \"https://svgomg.firebaseapp.com\"}
        }]
    </string>
</resources>
```

Note: the backslashes are intentional!

Deploy your assetlinks.json to prove ownership of the domain

To launch the site, you'll need to add a "statement" to the `.well-known/assetlinks.json` file with the package name and hash of your app's certificate. This allows Digital Asset Links to establish a bidirectional relationship between your site content and the app opening it in a Trusted Web activity (only you can generate the cert fingerprint and only you can host this file).

Note: you can get the hash of your app's certificate with the following command: `keytool -exportcert -list -v -alias androiddebugkey -keystore ~/.android/debug.keystore`

Creating a statement is documented here. An example statement for the sample app is:

```
[{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target": {
    "namespace": "android_app",
    "package_name": "org.chromium.twa.svgomg",
    "sha256_cert_fingerprints": [
      "82:04:C5:DB:19:A8:B9:8A:27:14:F0:3E:F5:23:2C:6B:B6:B9:63:10:F2:F9:CD:44:72
      "91:45:8F:34:E3:13:E4:58:1C:12:21:7A:FD:1E:BD:5C:BE:9B:DE:2C:1E:57:DC:0D:2B
    ]
  }
}]
```

You can verify the assertion on your site with the validator.

Troubleshooting:

- "Invalid input field(s)"
  - Specify the domain only (no https:// or path after the domain name)
  - Test only one signature at a time (the SHA256 hash should contain 32 octets, or pairs of hexadecimals, in uppercase and separated by colons.)
- "No app deep linking permission found": your site's **`.well-known/assetlinks.json`** file is inaccessible (try navigating to it to make sure you see the statements you wrote.)

Create an Activity that launches the Trusted Web activity

You can use TwaSessionHelper to instantiate and launch a Trusted Web activity from a new activity of your own, following TwaLauncherActivity in the sample.

Inside the **`TwaLauncherActivity`**, we use the **`onCreate`** event to ask the **`TwaSessionHelper`** to establish a connection to the **`CustomTabsService`** behind the scenes by calling the **`bindService`** method.

We also set the **`TwaLauncherActivity`** as a listener for the session events, so we can control what to do once a connection to the browser is established.

The first parameter for **`bindService`** is the **`Context`** to which the service will be bound to. The second parameter is an **`URI`** to the origin that we want to validate for the Trusted Web activity.

```
Uri originUri = Uri.parse(TWA_ORIGIN);
TwaSessionHelper twaSessionHelper = TwaSessionHelper.getInstance();
twaSessionHelper.setTwaSessionCallback(this);
twaSessionHelper.bindService(this, originUri);
```

When the relationship has been validated and the browser has been warmed up, **onTwaSessionReady** is invoked, to allow the **TwaLauncherActivity** to know that the Trusted Web activity is ready to be opened.

```
TwaSessionCallback twaSessionCallback = mTwaSessionCallback.get();
if (twaSessionCallback != null) twaSessionCallback.onTwaSessionReady();
```

The **onTwaSessionReady** callback tells us that the we are ready to open the page. In the context of this sample, we want to open it as soon as the session is ready, but other applications may want to open the Trusted Web activity as a result of a user action, and this event can be used to enable that action (eg: enable a button.)

Opening the Trusted Web activity is implemented in the **openTwa** method. The API allows us to pass our own **CustomTabsIntent** so we can change start and exit animations. Customizing animations is mostly relevant for applications opening Trusted Web activities from an existing activity.

```
// Set an empty transition from TwaLauncherActivity to the splash screen.
CustomTabsIntent customTabsIntent = twaSessionHelper.createIntentBuilder()
        .setStartAnimations(this, 0, 0)
        .build();
Uri openUri = Uri.parse(TARGET_URL);
twaSessionHelper.openTwa(this, customTabsIntent, openUri);
```

At this point, you should be able to load your app and open your test site!

We hope that you can experiment with this API and give us feedback at @ChromiumDev

---

Last updated June 28, 2018.