Detect if your Native app is installed from your web site

By Paul Kinlan
Paul is a Developer Advocate

As the capabilities of the Web become more aligned with what was once the domain of native experiences there are an increasing number of times where a developer will want to reduce the confusion for their users who have both the web and native apps installed.

Take notifications for example, introduced in Chrome 42; they allow developers to easily re-engage with users who opt to receive messages. But what if the user also has their native app installed? There was no way for you as the developer to know if your user has your app installed on their current device. If the user has the app installed there might be no reason to prompt for notifications from the web as well.

In Chrome 59 we are introducing a new API called `getInstalledRelatedApps()`. This new API lets you determine if your native app is installed on a device.

Note: This API is behind an Origin Trial, this means that we are in an experimental mode and are actively looking for feedback. You have to opt your site into this trial because it is not available broadly on the web. You can now sign up for the trial.

This is an incredibly powerful API because it gives you access to information that you can't infer from the web. This means that there must be a provable bi-directional relationship between your site and your native app. There are three core components that make this work.

1. There is a reference to your native app from your Web App Manifest via the related_applications property. This is your site saying that it is related to the native app.
2. There is a native app installed with the same package name as the one referenced in your Web App Manifest. The app must have a reference from your `AndroidManifest.xml` via the `asset_statements` element. This asserts that your native app has a relationship with your site.
3. If the above two criteria are met then a call to `getInstalledRelatedApps()` will resolve the list of apps.

These three steps are in place to ensure that only you can query your apps and that you have reliably demonstrated ownership of the site and app. They are now described in more detail.

Define the relationship to your native app in your Web App Manifest

You need to ensure that you have a Web App Manifest linked to from your site. In the manifest you must define a related_applications property that contains a list of the apps that you want to detect. The related_applications property is an array of objects that contain the platform on which the app is hosted and the unique identifier for your app on that platform.

```
{
  ...
  "related_applications": [{
    "platform": "play",
    "id": "<package-name>"
  }],
  ...
}
```

Note: Only Chrome on Android supports this, so the platform must be set to "play". You also need your "id" to be the exact package name for your Android App.

Create the relationship to your site in your AndroidManifest.xml

Next, you need to have your native app signal to the device that it is related to your Web App. You need to define the relationship with your site by ensuring that the app has the same package name as that defined in the Web App Manifest and that it also refers back to the website using the Android Digital Asset Links infrastructure.

Creating the link back to your site is possible by adding the following to your **`AndroidManifest.xml`**:

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

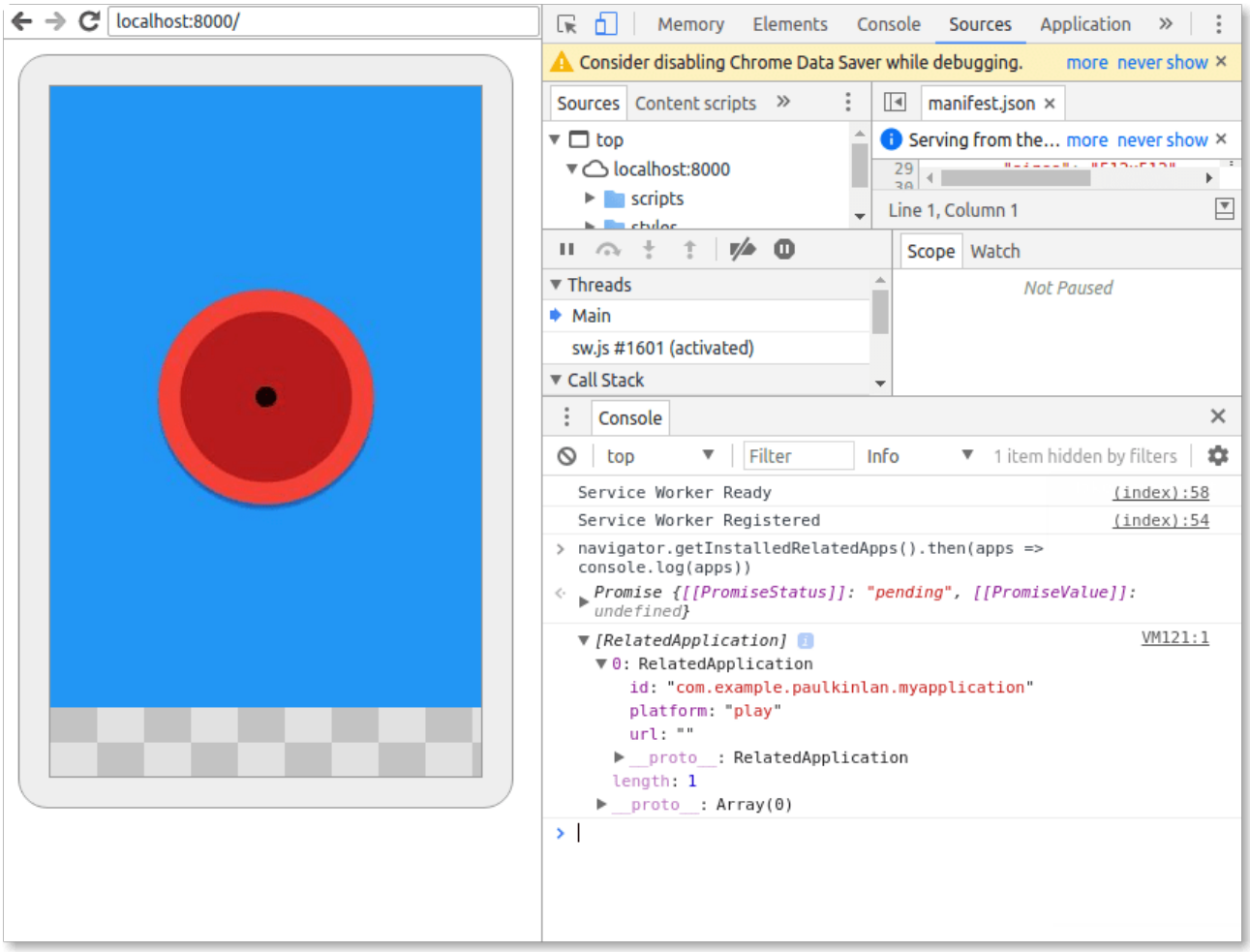
And then adding the following to your **`strings.xml`** resource, replacing the **`<site-domain>`** with the domain of your site:

```
<string name="asset_statements">
  [{
    \"relation\": [\"delegate_permission/common.handle_all_urls\"],
    \"target\": {
      \"namespace\": \"web\",
      \"site\": \"https://<site-domain>\"
    }
  }]
</string>
```

Test for presence of the app

Once you have the required metadata deployed on your app and on your site, you should be able to call **`navigator.getInstalledRelatedApps()`**. This returns a promise that resolves to the list of apps that are installed on the user's device that meet the above criteria (i.e., have been proved to be owned by the app developer).

```
navigator.getInstalledRelatedApps().then(relatedApps => {
  for (let app of relatedApps) {
    console.log(app.platform);
    console.log(app.url);
    console.log(app.id);
  }
});
```



Demo of Related Apps in DevTools

Testing on localhost

In your **`strings.xml`** set the "site" property value to be **`http://localhost:[yourportnumber]`** like the following.

```
<string name="asset_statements">
  [{
    \"relation\": [\"delegate_permission/common.handle_all_urls\"],
    \"target\": {
      \"namespace\": \"web\",
      \"site\": \"http://localhost:8000\"
    }
  }]
</string>
```

Ensure that your Web App Manifest has the correct package name for your locally installed App. When you deploy your Android App, make sure that you update the site property value to be the correct URL for your site. The most efficient way to manage this is through your product flavours (Release, Debug, etc.) which allows you to specify different resource files

based on the build target, meaning that your Release target will only ever contain your live domain.

Use cases

There are many different ways that you can use this API. Here are some quick examples of possible uses and common pieces of functionality that you will find useful.

Cancel the progressive web app installation if the native app is installed

You can intercept the **`beforeinstallprompt`** event. The general flow is to call **`preventDefault()`** on the event so the banner doesn't show right away, check to see if there are no apps installed and if so call **`prompt()`** to show the banner.

```
window.addEventListener("beforeinstallprompt", e => {
  if (navigator.getInstalledRelatedApps) {
    e.preventDefault();  // Stop automated install prompt.
    navigator.getInstalledRelatedApps().then(relatedApps => {
      if (relatedApps.length == 0) {
        e.prompt();
      }
    });
  }
});
```

Prevent duplicate notifications

One of the intended uses for this API is to allow you to de-dupe notifications from both your Native Application and your Web App. There are a couple of options available to you.

If the user has your application installed and they don't have notifications enabled already on your site, you can use the **`getInstalledRelatedApps()`** method to selectively disable the UI that the user would normally use to enable the feature.

```
window.addEventListener("load", e => {
  if (navigator.getInstalledRelatedApps) {
    navigator.getInstalledRelatedApps()
    .then(apps => {
      if(apps.length > 0) { /* Hide the UI */ }
    });
```

```
  }
});
```

Alternatively, if the user has already been to your site and has Web Push enabled you can unregister the push subscription during the onload event.

```
window.addEventListener("load", e => {
  if (navigator.getInstalledRelatedApps) {
    let sw = navigator.serviceWorker.ready;

    navigator.getInstalledRelatedApps()
    .then(apps => (apps.length > 0) ? sw.then(reg => reg.pushManager) : undefined
    .then(pushManager => {
      if(pushManager) pushManager.unsubscribe();
    });
  }
});
```

This API is not available directly to the service worker so it is not possible to de-duplicate notifications as they arrive at your service worker.

Detect if your PWA is installed from a native app

If the user has installed your Progressive Web App through the old Add to Homescreen method (i.e, in anything prior to Chrome 58) then it is not possible to detect if your app is installed. Chrome added your site to the Homescreen as a bookmark, and this data was not exposed to the system.

If the user has installed the web app using the new Web APK functionality, it is possible to determine if your web app is installed. If you know the package name of your Web APK then you can use the `context.getPackageManager().getApplicationInfo()` API to determine if it is installed. Please note that this is experimental.

Not Working?

File a bug right here against the Chrome implementation. The correct people will be notified (I've sneakily put this in, so I am sure they will be grateful).

We are keen to keep getting feedback on the spec and if you have any issues or suggestions, file an issue against the spec

Last updated July 2, 2018.