PWAs as URL Handlers

This Document is Public

Authors: lu.huang@microsoft.com, mandy.chen@microsoft.com
Contributors/Reviewers: <your_email_here@>
Last Updated: March 2021

Status: In development, M91 Dev Trial

Explainer

WICG/pwa-url-handler

Platforms

Desktop platforms (Windows, MacOS, Linux)

Tracking Bug

1072058 - Ability for PWAs to be registered as URL Handlers

Implementation Status

See crbug for merged CLs. See tracking sheet for in progress and planned CLs. See Chrome status page for dev trial, origin trial, and launch status.

Summary

Developers can create a more engaging experience if Progressive Web Apps (PWAs) are able to register as handlers for https uniform resource locators (URLs). This explainer proposes a scheme for a PWA to register as a URL handler with the browser during installation, and launch when relevant URL links are activated. This document describes the design for implementing the feature in Chromium.

Background

Today, native applications on many operating systems (Windows, Android, iOS, macOS) can be associated with http(s) URLs. They can request to be launched as URL handlers when associated URL links are activated. For example, a user could click on a web link to a news story from their native email client. An associated native app for viewing news stories that the user has installed could be launched directly to handle the activation of the web link. PWAs would be more comparable to native apps if they also had the ability to declaratively register as URL handlers from within their web app manifest.

There was a past experiment in Chromium that captured link navigations and launched PWA windows if the link URL was in the PWA's app scope. This work informed sw-launch, a more recent proposal which gives PWA developers more control over how URLs in scope of an installed PWA are launched by triggering an event in the service worker. Recently (May 2020), sw-launch added a declarative link capturing (DLC) design that makes use of the web app manifest as well. A declarative approach allows for validation at installation time instead of navigation time, thereby enabling integration with OS URL handling features. URL handling at the OS level is an important goal for us and would further improve the PWA experience with native UX.

While both declarative link capturing and this proposal would allow a PWA to launch in an app window in response to a URL launch, DLC only targets in-scope URLs. URL Handling would allow out-of-scope URLs to also be captured by PWAs. From discussion with DLC's authors, we decided that the two proposals can be developed independently, with one focused on the launch behavior for in-scope URLs, and the other focused on enabling PWAs to handle URLs from different origins.

Goals

Refer to the explainer for the latest set of goals. This document describes the dev. design for browser level URL handling in Chromium. Components touched include the blink web app manifest parser, web app database, OS integration manager, local state prefs, and startup browser creator. We also describe components for web app to origin association validation in this document.

Non-Goals

This document does not discuss custom URL protocol handling, which being designed as a separate feature. There is a separate explainer and dev design spec for that feature.

This document does not discuss the registration of URL handlers with the OS. We will start a separate design doc in the future for OS integration.

Synopsis

The "Functional Design" section serves as feature specification. Skip ahead to "Interfaces and Interactions" and "Dev Design" for technical design.

1. Functional Design

First, we describe the different usage scenarios of URL handling as well as expected behavior. Not all details in the explainer are duplicated here.

1.1. PWA developer

PWA developers will be able to include a **url_handlers** member in the web app manifest. This member contains web origin strings. The installed app can be launched by the browser to handle incoming URL link activations if the URL matches any of those origins. PWAs can handle URLs that are outside of their own app scope and even from other origins.

The out-of-scope content needs to participate in a handshake with each app they recognize. We propose a web-app-origin-association file format that origins could use for the handshake. The browser will validate this handshake.

Content owners are able to configure the allowed and disallowed paths on their origin for each web app they recognize for this feature.

1.2. PWA user

When a user installs a PWA with an **url_handlers** manifest member, the browser will silently process its contents and register that PWA for URL handling. The user does not have to give consent at install time.

When the user activates a link outside the browser and the browser is launched, all PWAs registered for URL handling from all profiles will be matched against the launch URL.

- OSes commonly carry out a URL activation request from native applications by launching the default http/https protocol handler app (usually a browser) with a URL parameter. (This means that our current design must work through the default chrome.exe startup process instead of a special-purpose executable, package, or shim.)

If no apps match the launch URL, browser startup will proceed normally and likely result in the URL being rendered in a new window or tab.

When one or more matches are found, what the user experiences depends on these factors:

- Is there a previously saved user choice for the *(app_id, profile, url_handler entry)* combination that matched?
- Are there more than one *(app_id, profile, url_handler entry)* combination that matches the input URL? If so, has the user saved a preference for one of them in the past?

1.3. User intent, User choice

Instead of displaying the url_handlers information to the user and asking if they give permission to activate the registration (i.e. an install-time permission model), newly registered url_handlers will be registered silently. Before a matching app is launched, the user needs to provide an explicit action to indicate their permission. This is collected with an intent picker dialog.

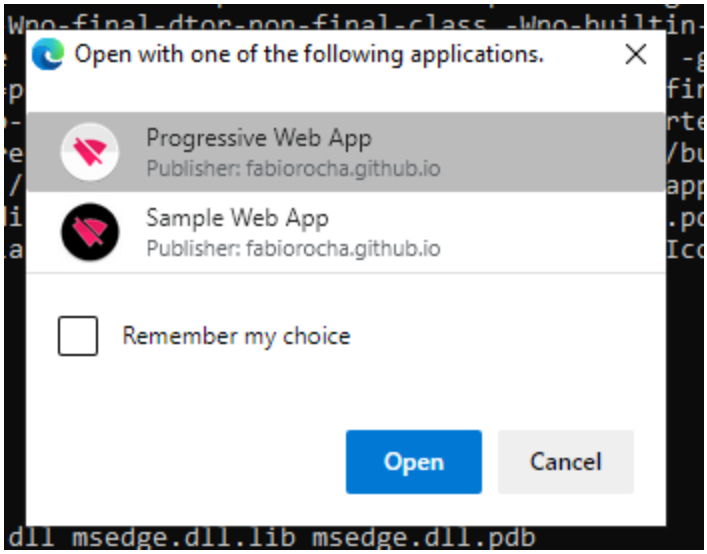

Figure 1: Intent Picker Dialog mockup.

This dialog is shown every time unless the user has previously launched an app and checked "Remember my choice".

~~1.3.1. Permission~~

~~Permission is the implied consent given by the user by performing the action of selecting an app and clicking ok. Permission is recorded at the *(app_id, profile, url_handlers entry)* granularity.~~

- ~~The *(app_id, profile, url_handlers entry)* combination will receive permission if that app_id and profile is launched from the intent picker.~~
- ~~Permission granted to a *(Profile A, App1, "https://app1.com/apple")* combination does not carry over to *(ProfileA, App1, "https://app1.com/orange").*~~
- ~~Recording permission at *(app_id, profile)* granularity was considered but rejected. Because an app can request to handle URLs with completely different origins, users should grant permissions to individual url_handlers entries separately to avoid confusion.~~
- ~~If multiple url_handlers entries from the same app and profile matched, they all receive permission. (The user picks an app, not a url_handlers entry.)~~
- ~~If the dialog is closed or canceled without launching an app, no combination receives permission. URL is opened in a tab.~~

~~The "Remember my choice" checkbox does not have to be checked to save permission values. The app has to be selected and launched.~~

1.3.2. Choice

Because the intent picker is displayed whenever there are 1 or more apps that match the input URL, we allow users to save their app choice with a checkbox. The intent picker dialog has a "Remember my choice" checkbox to allow the user to save their selection of matching apps if a similar URL is observed in the future.

In similar designs, the user is often allowed to save the user's choice of app for handling a particular URL protocol or URL origin, etc. The Apps For Websites feature in Windows 10 uses origin as the key for saving the user's choice of apps associated with a URL. Eg. If two separate apps FooApp and BarApp both use a free host, apps.com, they could both register for AppsForWebsites URL handling. If they are hosted at apps.com/foo-app and apps.com/bar-app respectively, they could register to handle their own respective paths. An input URL will not match both apps unless both apps register to handle "apps.com/*". Otherwise they can each be saved as the default choice for apps.com without being presented side by side. If they are hosted at foo-app.apps.com and bar-app.apps.com respectively, they could also each independently be saved as the default app.

PWAs can also save the user's app choice using the above-mentioned model. Even though PWAs are naturally associated with a path instead of an entire origin, it is still possible to save the default choice and key it by origin. PWAs could be organized under a sub-domain, e.g. [app name].host.com to make them different origins.

When a URL (e.g. not-a-pwa.com/some-content) is matched to one or more PWAs, we want to know if there are any among them that have been saved as the default app for the origin: *not-a-pwa.com*. **It is possible we could find more than one:** App1 registers for not-a-pwa.com/sub-path/* -> user activates third-party-not-a-pwa.com/sub-path/some-content -> user selects App1 and checks remember my choice. App2 registers for not-a-pwa.com/ * -> user activates not-a-pwa.com/some-content -> App1 is not displayed as an option because it is more restrictive -> user selects App2 and checks remember my choice. When the user activates not-a-pwa.com/sub-path/some-content again, **both App1 and App2** will be presented as options, and both have been saved as the default handler for the origin not-a-pwa.com. Nested scopes could also have the same problem.

~~Since there is a tie, we will present both options even if the options may not make sense to the user as alternatives. Checking the checkbox during this selection would not have any effect so it will be hidden instead. The user will **always have to pick** between these two apps for that URL.~~ This behavior is not ideal but we believe it will rarely be exposed to the user if apps are conscientious about using sub-domains and writing good url_handlers.

Once saved, default choices will have to be changed by the user by visiting chrome://settings/content/urlHandler. Because saved choices are keyed by origin, they can be displayed in groups by origin. Saved choices can be removed with a button.

~~We want to launch a PWA as a default choice only if it matches a *(app_id, profile, url_handlers entry)* combination and an origin *: (app_id, profile)* entry is found in the saved decisions. This means that different url_handlers entries of the same app will share a saved decision if they have the same origin, but will not otherwise. In the most common case where all of an app's url_handlers entries target in-scope URLs, saving it as a choice will always skip the intent picker for future matches. This design allows the user to mentally associate apps with origins instead of having to understand PWA scope.~~

1.4. App Launch

When an *(app_id, profile)* has been found as the matching handler for an input URL, it can then be launched without user input in some cases, or using the intent picker dialog with user input. What URL is navigated to will depend on whether the input URL is within the app scope.

1.5. App Launch With Out-of-Scope URLs

How out-of-scope URLs should best be presented within an app window if they cause an app launch through URL handling is still being developed. Our current design is to forward information after a matching app is selected to a document event handler and allow it to control the launch behavior. This is similar to the existing_client_event option from declarative link capturing and is intended to be forward compatible.

Other options for handling out-of-scope URLs:

- App launches to its start URL.
- Render the out-of-scope URL in the app window within CCT UI.
- Pass launch information to the start_url document using URL parameters.
- Launch information and control could be handled to a service worker event handler.

1.6. App Installation Behavior

Other aspects of the app installation behavior are unaffected. In the future, **url_handlers** in the web app manifest can registered with the OS using OS-specific URL handling APIs (eg. Windows.appUriHandler).

1.7. App Uninstallation Behavior

App uninstall behavior is unaffected. All app data associated with a particular app install will be removed during app uninstall, including its URL handling registrations and preferences.

1.8. Manifest Update

When a PWA undergoes web app manifest update, it is possible for the **url_handlers** data to change. A manifest update will cause origin associations to be revalidated. New or modified **url_handlers** entries will update registrations. Saved app choices that are affected will also be cleared.

1.9. Profile deletion behavior

If a profile is deleted, all installed app data from that profile, as well as their associated browser preferences will be removed. Any URL handling registrations will also be removed.

1.10. Settings available in the settings page

Similar to how there is a selection for protocol handler settings at chrome://settings/content, we will add a selection for URL handler settings.

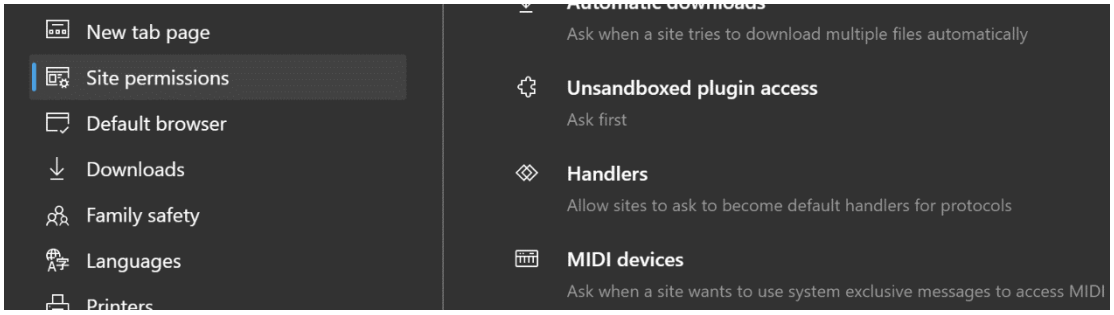

Figure 3 We will place "URL Handlers" here and rename "Handlers" to "Protocol Handlers".

Similar to the protocol handlers settings page at chrome://settings/handlers, we will add a settings page at chrome://settings/urlHandlers to manage permissions and preferences for URL handlers.

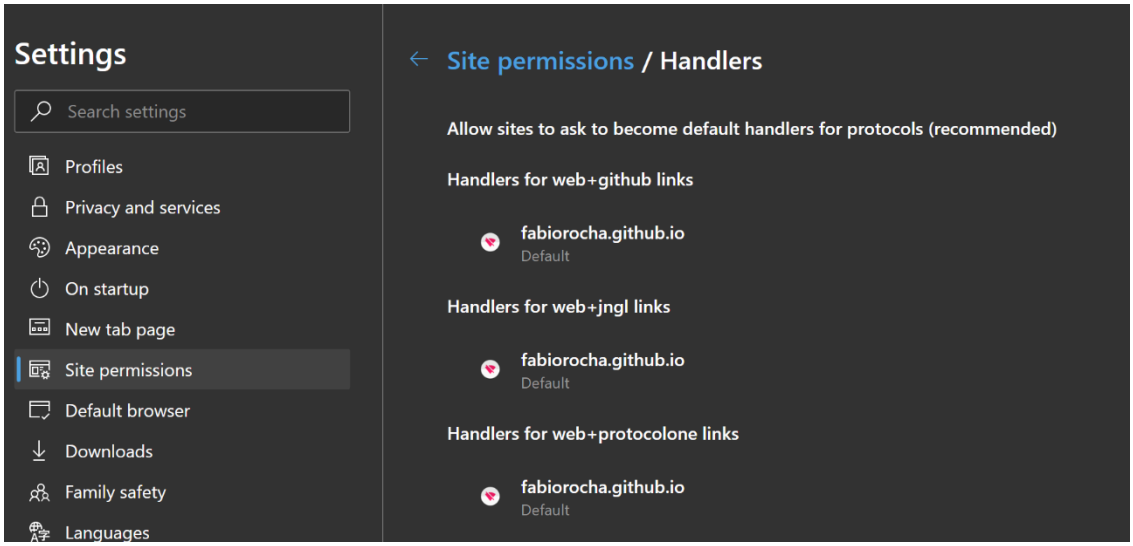

Figure 4: Protocol Handlers settings page. We will create a similar page to control saved app choices for URL handlers.

1.11. Notifications

If necessary, URL handling actions can be explained to the user using notifications. We will show a notification from the app window if URL handling successfully launches an app. This could be implemented as an infobar, with a link to a "Learn more" page, and another link to chrome://settings/content/urlHandlers.

"Chrome launched www.app.contoso.com/... in [app icon] [app name]. [Learn more] [Manage]"

This infobar will keep showing for a user profile until the user interacts with it (by clicking on the links or closing it).

2. Interfaces and Interactions

2.1. Public API Added/Changed

This design adds a new manifest member, **url_handlers**, to the web app manifest specification and introduces a new **web-app-origin-association** file format.

- **url_handlers** allow PWAs to declaratively register as URL handlers for a set of URLs.
- The **web-app-origin-association** file allows origins to grant permission to be associated with PWAs and for their URLs to be handled by those PWAs.

Refer to the explainer for examples of **url_handlers** and **web-app-origin-association**.

2.2. API Consumption

The new web app manifest field **url_handlers** can be set by PWA developers to register a PWA as a URL handler.

The new **web-app-origin-association** file is consumed by owners of origins specified in **url_handlers**. The content owner must host a **web-app-origin-association** file to validate the association with the PWA. The explainer specifies where the **web-app-origin-association** file must be hosted.

2.3. Breaking Changes

There are no breaking changes as a new web app manifest member is added. Changes in the web app manifest as well as the **web-app-origin-association** file will be ignored by browsers that do not implement this design (backwards compatible). All new features will be gated by a feature flag. Activation of these features does not require any data migrations or conversions. The new feature flag will be named **enable-desktop-pwas-url-handling**. We may also create a runtime feature flag later to enable origin trial.

2.4. Privacy and Security Considerations

Refer to the explainer for privacy and security considerations. Also see the privacy and security self-review.

3. Dev Design

3.1. Overview

Add **url_handlers** member to the web app manifest

PWAs register for URL handling by adding the **url_handlers** member to their web app manifest. This data is parsed from the manifest, transferred from the render process to the browser process, converted to web_app::WebApp objects and serialized to a protobuf database. Manifest content is accessible from web_app::WebAppRegistrar.

Validate **web-app-origin-association**

If **url_handlers** are present and contain origin strings, the browser needs to validate the web-app-origin-association files of those origins to get their content

Build an index of URLs and apps registered for URL handling

In addition to parsing and storing the **url_handlers** data, the browser will maintain an index that is updated with new URL handling registrations for efficiently matching URLs with handlers. The index will be backed by local state prefs, and it will be updated by installations, uninstallations, and updates of PWAs from all profiles as well as relevant profile and web app registrar changes. This enables any installed app from any profile to be launched at browser startup if there is a match.

Match input URL to one or more URL handlers. Launch app.

If the browser is started with a command line https URL argument, that URL will be matched against registered URL handlers. Results will be displayed by an intent picker dialog.

Intent picker dialog, notification, settings page, DevTools

We will also implement user settings features and developer tools features:

- Show an intent picker dialog so users can control app launch with an explicit action.
- Show saved default URL handlers in chrome://settings/content/urlHandlers page.
- Show a notification when an app is launched.
- F12 DevTools Application panel shows information about URL handlers to aid development.

3.2. Manifest parsing and storage

The new **url_handlers** member is parsed from the web app manifest in the renderer process, sent to the browser process, stored in a WebApp object, and is then serialized in proto format for storage to the web app database. We modify the manifest parser, the mojom IPC from renderer to browser process, the web_app::WebApp class, and the web_app::WebAppDatabase class that serializes/deserializes WebApp objects to disk.

3.2.1. Manifest Parser Changes

/third_party/blink/renderer/modules/manifest/manifest_parser.cc
We add new methods to parse the value of url_handlers. They will be called from within ManifestParser::Parse().

The manifest parser enforces limits on the number of url_handler entries per web app manifest (10), and limits on the lengths of each path and each exclude_path (2000 characters).

3.2.2. Mojom Changes

/third_party/blink/public/mojom/manifest/manifest.mojom
We add a field to the mojom struct definition for Manifest.

```
struct Manifest {
 array<ManifestUriHandler> url_handlers;
  ...
}
struct ManifestUrlHandler {
 mojo_base.mojom.String16 origin;
```

```
};
```

Similarly, we add equivalent manifest fields to:
/third_party/blink/public/common/manifest/manifest_mojom_traits.h
/third_party/blink/public/common/manifest/manifest.h
/third_party/blink/renderer/modules/manifest/manifest_type_converters.h/cc

/third_party/blink/common/manifest/manifest_mojom_traits.cc
Modify StructTraits::Read to account for url_handlers.

3.2.3. WebAppInfo, WebApp

/chrome/common/web_application_info.h
Add url_handlers to WebApplicationInfo.

```
std::vector<blink::Manifest::UriHandler> url_handlers;
```

/chrome/browser/web_applications/web_app.h/cc
Add an apps::UrlHandlers member, getter and setter to web_app::WebApp. Expose this information through the web app registrar.

3.2.4. Manifest Histogram

/third_party/blink/renderer/modules/manifest/manifest_uma_util.cc
There are existing histograms for each of the manifest fields. Add a new histogram for url_handlers.

3.2.5. Other Changes

/third_party/blink/common/manifest/manifest.cc
Edit Manifest::IsEmpty to check for url_handlers.

/chrome/browser/web_applications/components/web_app_install_utils.cc
Copy manifest.url_handlers in UpdateWebAppInfoFromManifest to WebApplicationInfo.

3.3. URL handler Registration and Indexing

We want to avoid matching for apps by iterating all installed apps from all profiles. Therefore, during installation, we also add to an index of active URL handlers during the web app installation. This index is built using local state prefs. A new class *UrlHandlerPrefs* is added to manage these browser prefs.

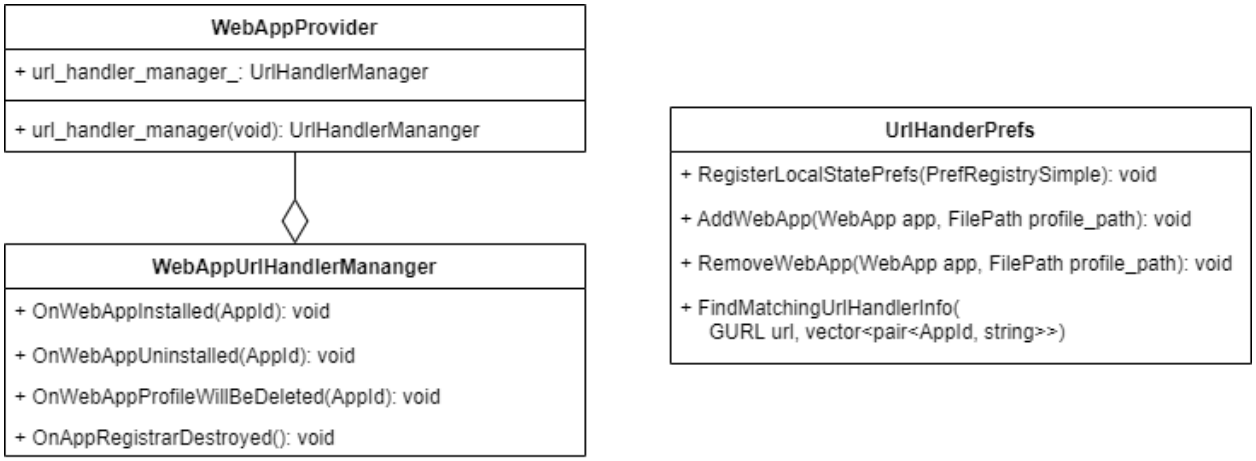

Figure 5: new classes added to manage an index of URL handlers

A new class *WebAppUrlHandlerManager* is added to handle web app life-cycle events. *WebAppUrlHandlerManager* will update URL handler registrations by observing app installations, un-installations, and updates. It does so by inheriting from and implementing web_app::AppRegistrarObserver.

WebAppUrlHandlerManager also initiates **web-app-origin-association** validation during app installs when it finds out-of-scope URLs in url_handlers.

3.3.1. UrlHandlerManager

/chrome/browser/web_applications/components/url_handler_manager.h/cc
UrlHandlerManager is a subsystem of WebAppProvider and it implements AppRegistrarObserver. It will update URL handler information using UrlHandlerPrefs in its implementations of OnWebAppInstalled, OnWebAppUninstalled, OnWebAppProfileWillBeDeleted, and OnAppRegistrarDestroyed.

UrlHandlerMananger is pure virtual and will be implemented by the class WebAppUrlHandlerManager. We will not implement this for bookmark apps.

3.3.2. Browser Preference

A new local state preference, kWebAppsUrlHandlerInfo, is added to store URL handler information. It will be in the following format:

```
{
 origin: [
       [app_id, profile_path, {handler_1}],
       [app_id, profile_path, {handler_2}],
       ...
 ]
}
```

The profile path is stored to support handling URLs by PWAs installed in any profile. An app installed with different profiles has the same app id in all the installed profiles, which makes storing the installed profile path required to distinguish apps installed in multiple profiles.

The handler dictionary structure mirrors the handler format in the manifest.

```
{
 "web_apps": {
    "url_handler_info": {
      "https://luhuangmsft.github.io": [ {                                    // origin key
        "app_id": "jncifgjpfigpfjphlanoeonmiedopibl",
        "exclude_paths": [  ],
        "has_origin_wildcard": false,
        "paths": [  ],
        "profile_path": "C:\\Users\\luhua\\AppData\\Local\\Google\\Chrome SxS\\User Data\\Default"
      } ]
    }
  }
}
```

3.4. URL matching

Given an input URL, UrlHandlerPrefs needs to be able to match it against the index of installed PWAs and their corresponding profiles.

3.4.1. UrlHandlerPrefs

UrlHandlerPrefs has the following interface:

```
class UrlHandlerPrefs {

 // Register prefs associated with the URL handling in local state
 static void RegisterLocalStatePrefs(PrefRegistrySimple* registry);

 // Add the associated URL handlers along with the app id and installed profile path to browser
prefs.
 static void AddWebApp(const WebApp* web_app, const base::FilePath profile_path);

 // Remove the associated URL handlers from the browser prefs.
 static void RemoveWebApp(const WebApp* web_app, const base::FilePath profile_path);

 struct MatchingHandlersResult {
   AppId app_id;
   std::string profile_path;
 };
 // Find the apps and their installed profiles that can handle the url.
 static std::vector<MatchinghandlersResult> FindMatchingHandlers(const GURL& url);
};
```

AddWebApp is called in ~~OnWebAppInstalled~~ OsIntegrationManager::InstallOsHooks(...) to add the handlers associated with the app installed to browser prefs. Likewise, RemoveApp is called in OnWebAppUninstalled.

To find matching handlers given an input URL, we can search the browser prefs for the origin of the URL.

“origin” values containing “*.” wildcard prefixes can also be indexed by excluding the prefix. This allows for different subdomains to be matched using one string.

3.5. web-app-origin-association file

Please refer to the Web App Origin Association design doc.

3.6. Browser Startup

Where the main browser process is started, existing code in StartupBrowserCreator extracts URL parameters from the command line. This is where we match the URL input against registered handlers. By doing it here, we are able to find matches before any resources like Browser or WebContents are created. Importantly, this happens before displaying any part of the browser UI. This allows PWA launches to appear naturally without first blinking the browser window.

All the information we need to do the URL-to-PWA matching is available from the browser process’s local state prefs, which will be loaded and readable when we perform the search. We call static functions in UrlHandlerPrefs to search for matching apps and profiles.

There is already a Profile loaded at this point. If no matches are found, browser startup continues normally. If an app is matched and chosen for launch, we will load and launch it with the installing profile. Note that two different profiles that each have the same app id installed may have different app manifests and contents stored due to factors like manifest update.

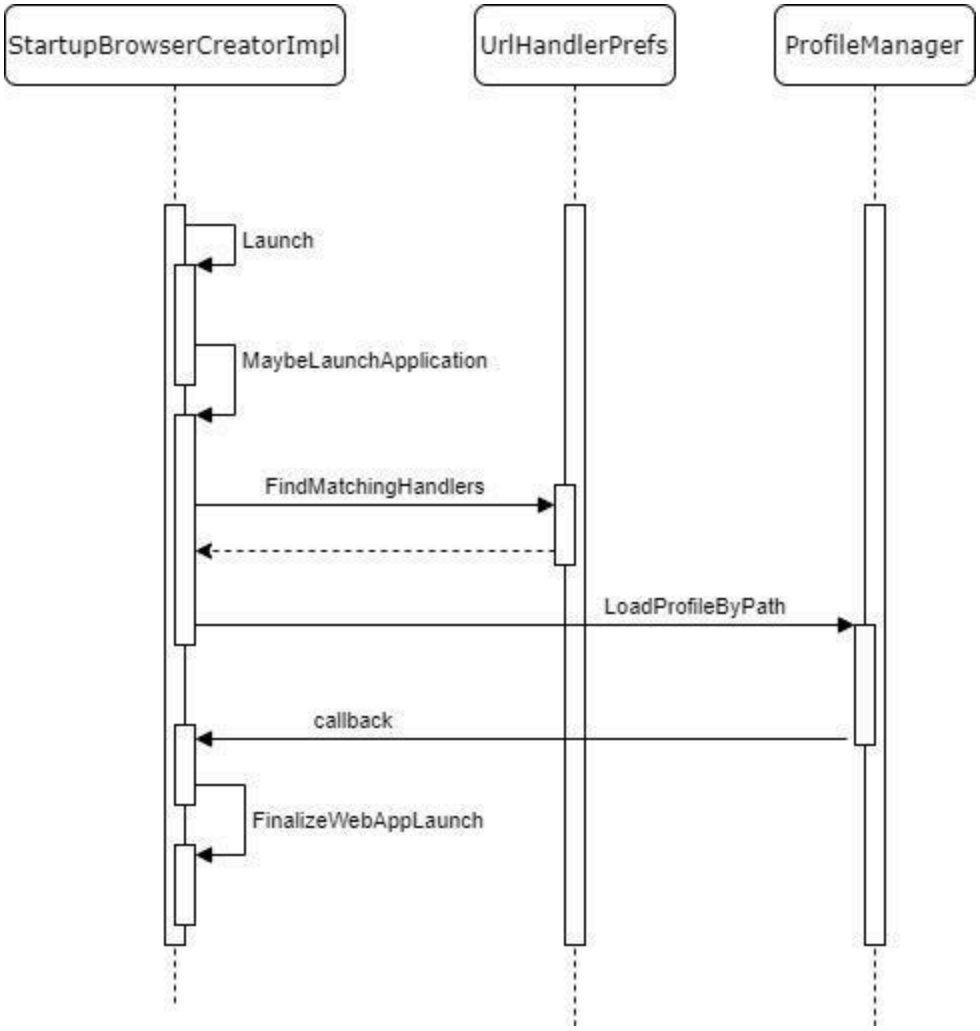

Figure 7 sequence diagram to search and launch app

3.6.1. Startup Location

/chrome/browser/ui/startup/startup_browser_creator_impl.cc
We modify StartupBrowserCreatorImpl::Launch to find matching URL handlers through UrlHandlerPrefs if the browser is launched with a URL param. StartupBrowserCreatorImpl::Launch is executed at browser startup, before any UI is created. If there are no matches, the browser startup process will continue as usual.

If there are matches, we will display an intent picker dialog for the user to choose which app should handle the URL. If the chosen app is installed in the currently loaded profile, we will proceed to launch the app. If it is installed in a different profile, we will load the target profile through ProfileManager::LoadProfileByPath and launch the app once the profile is loaded.

3.6.2. Startup Performance

If the feature flag is disabled, no computation or resource allocation is added to startup.

If the feature flag is enabled but there are no URL handlers registered (there are no PWAs installed for any profile or no installed PWAs have url_handlers declared in their manifests), there will be no data in the index in local state prefs and no further computation.

We expect that most PWAs available in the world will be associated with a single origin: their own. (eg. unique-app.com, or app-one-of-many.apphost.com). Searching for matches is efficient because most probably the origin of the URL will key to the url_handlers entries from a single app and from a single profile.

It is possible for apps that share an origin to be organized by path instead of sub-domain. Eg. host.com/app1 instead of app1.host.com. It is possible for many apps to be keyed by the same origin. It is unlikely that sites like this will have a large number of apps with **url_handlers** and that users will install many of them at the same time.

3.6.3. Protecting Startup Performance

To ensure reasonable worst-case performance:

- We will limit the maximum number of url_handlers entries each web app manifest can register to 10.
- The paths and exclude_paths in each url_handlers entry will also be limited to 10.
- We will limit the length of each “origin” or “paths” entry, or “exclude_paths” entry to 2000 characters.

Using the dictionary type browser prefs to storage handler information should ensure the “origin” portion to be matched in linear time. Because of the possible wildcard prefix in “origin”, the URL input might have to be trimmed and searched for more than once. Once a matching “origin” is found, matching the “paths” and “exclude_paths” strings should be in O(m.n) time, where m is the number of “paths” and “exclude_paths” strings and n is the length of the URL path substring. We will test URL matching with pathological input.

3.6.4. Intent Picker Dialog (disambiguation dialog)

If more than one registered URL handler is found for a URL query, present an intent picker dialog to the user so they are able to choose which handler to launch. This dialog is similar in appearance to the “Open with” dialog shown when the “To open this link, choose an app” icon in the omnibox is clicked.

The design and development of this dialog is shared with other features teams, as the dialog will be also used by features such as PWA protocol handling.

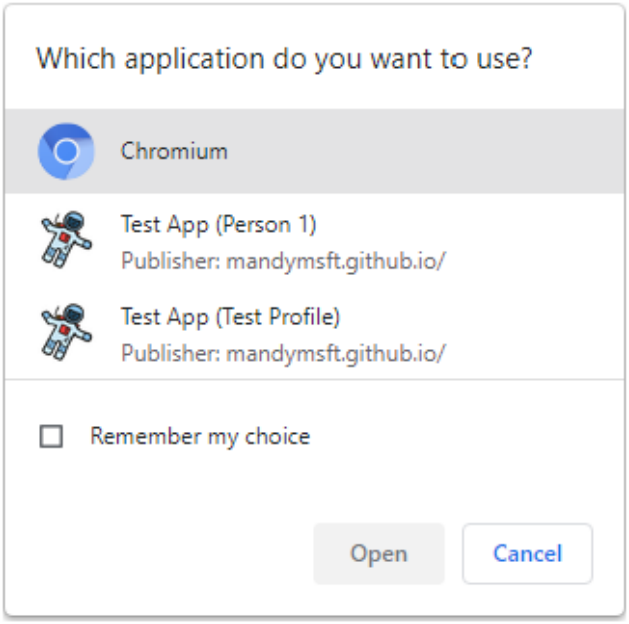

Figure 8: Intent Picker dialog for web app URL handling

Each matching app option would be displayed. ~~Closing or cancelling the dialog would open the URL in a new tab page.~~

See the UX Review doc for more discussions.

See the PWA URL Handlers Saved Choice Design for details on how “Remember my choice” is implemented.

3.7. Other User and Developer Experiences

URL handling information should be displayed appropriately for both users and developers.

3.7.1. Pop-up notification in app window after launch

Implement a dialog or pop-up UI that is displayed after a PWA is launched by the URL Handling feature. This UI would present a message that explains that the PWA was launched because it was registered as a URL handler. This UI should present the user with the option to dismiss the message (or dismiss itself after n seconds), or go to the settings page for URL handling.

3.7.2. chrome://settings

We will implement a settings page at chrome://settings/content/urlHandlers to allow the user to control URL handling saved defaults for installed apps.

3.7.3. chrome://apps

We could extend the chrome://apps page's app context menu to show a link to the chrome://settings/content/urlHandlers page.

3.7.4. DevTools

We could extend the Application pane in the F12 DevTools to display URL handling information for PWAs and allow PWA developers to debug and troubleshoot their manifest configuration and web-app-origin-association configuration..

- Display any **url_handlers** parsing errors
- Display **web-app-origin-association** validation errors.

4. Implementation plan

We will implement this design in the following steps:

1. Add feature flag *enable-desktop-pwas-external-url-handling* ✔️
2. Add manifest parsing and web app database changes (CL link) ✔️
3. Add WebAppProvider components and utilities ✔️
4. Add StartBrowserCreator components and utilities ✔️
5. ~~Add browser tests for cases that do not require association validation~~
6. Add association file fetching, parsing, and processing components and utilities ✔️
7. Add association validation logic to url_handler_manager ✔️
8. Add web-app-origin-association validation ✔️
9. Add Browser tests for cases that require association validation ✔️
10. Add implementation for handling out-of-scope URLs ✔️
11. Add telemetry
12. Add usage of the intent picker dialog
13. Add chrome://settings/content/urlHandlers
14. Add runtime feature flag for origin trial

5. Rollout Plan

Experiment-controlled rollout. Experiment name: TBD.
Milestones:

- M91 Dev Trial

    - Able to parse new manifest member
    - Able to fetch, parse, and validate web_app_origin_association file
    - Able to launch new PWA instance and navigate to launch URL
- M92 Origin Trial
    - Able to display matching PWAs in dialog and prompt user for action
    - Able to save user app choices
    - chrome://settings/content/urlHandlers available
- M9x Ship
    - Enable feature flag by default

6. Adapt to standardisation requirements

As the web standard incubation progresses, we will update this document to reflect necessary changes. We do not expect any significant design changes.

Currently proposing URL handling as a web standard here: https://github.com/WICG/pwa-url-handler/blob/master/explainer.md.

At the same time, Declarative Link Capturing for in-scope URLs is proposed here: https://github.com/WICG/sw-launch/blob/master/declarative_link_capturing.md

We are also trying to keep the include/exclude pattern syntax compatible with that proposed for scope matching here: https://github.com/wanderview/service-worker-scope-pattern-matching/blob/master/explainer.md

Current differences between the explainer and design doc:

- The explainer used to propose that the association file's URL is declared in the web app manifest. It now proposes that where the association file must be found will not be part of the spec and will be determined by the implementation. This doc is being updated to reflect this. We are experimenting with two different options: 1. A link rel in the PWA document, similar to how the web app manifest is found. 2. A fixed location relative to the origin being associated, similar to how native formats make use of the .well-known path.
- "url_handlers" as the name of the manifest member is changing. Currently the explainer uses the name "url_handlers".
- Some concepts in the explainer such as permissions and saved app choices in site settings are based on "site", "origin", "host", etc. These choices are still being discussed. This doc may at times be out of sync with the explainer on those issues.

7. Telemetry and Flighting

7.1. Telemetry

New telemetry may be needed for the following:

- **url_handlers** parsing failures
- **web-app-origin-association** download and parsing failures
- Usage of **url_handlers** and **web-app-origin-association** in PWAs
- In intent picker dialog, whether users choose a PWA or close or cancel dialog
- In intent picker dialog, whether users choose to set default choice (checkbox checked)
- When intent picker dialog presents two or more app choices
- New settings page usage: chrome://settings/urlHandlers

8. Functional and Unit Testing

8.1. Approach

Library components will be tested with unit tests. Larger components with dependencies will be tested with mocks/fakes or using browser tests.

8.2. Test Cases

- Test that the url_handlers manifest property member is parsed correctly and values are correctly plumbed through to WebApp structs and to proto storage.
- Test that the URL and UrlHandlerInfo matching library works correctly.
- Test that the Indexed storage of URL handler registrations works identically to the reference implementation.
- Test that the web-app-origin-association download and validation library works correctly.
- Test that the permissions and confirmation UI work correctly using browser tests
- Tests for any additions/modifications to DevTools
- Tests for any chrome://settings pages modified

8.2.1. Automated Test Cases

Individual components will have unit tests. We will try to use webdriver to create end-to-end tests of PWA URL handling scenarios.

8.2.2. Manual Test Cases

We will manually test these scenario:

- Install a PWA that has url_handlers, both in and out of app scope
- Activating URLs that match the PWA's url_handlers from the run dialog
- Activating URLs that do not match the PWA's url_handlers from the run dialog

- Activating the same URLs from links in a native app like Outlook/OneNote.

9. Performance Investigation

See performance investigation doc here.

10. Metrics

Success Metrics

Regression Metrics

Experiments

11. Open Questions

-

12. Links

Public explainer (archived)	https://github.com/MicrosoftEdge/MSEdgeExplainers/blob/master/PwaUriHandler/explainer.md
WICG explainer github	https://github.com/WICG/pwa-url-handler
WICG Discourse page	https://discourse.wicg.io/t/proposal-pwas-as-uri-handlers/
Intent picker UX design	1105257 - Browsers need a way to disambiguate between multiple registered PWAs at launch time for a specific URL/Protocol handler" - chromium
Privacy and security self-review	https://github.com/WICG/pwa-url-handler/blob/master/PRIVACY_AND_SECURITY.md
Chrome status	Progressive Web Apps as URL Handlers - Chrome Platform Status (chromestatus.com)
TAG review	https://github.com/w3ctag/design-reviews/issues/552
crbug	1072058 - Ability for PWAs to be registered as URL Handlers (chromium.org)
Performance impact evaluation	PWA URL Handling Performance Investigation
Saved choice design	PWA URL Handlers Saved Choice Design

Web app association design	Web App Origin Association Design - Google Docs