

PWA Navigation Capturing (public version)

This Document is Public

Authors: Daniel Murphy, Marijn Kruisselbrink, Dibyajyoti Pal, Robbie McElrath

Last Modified: Jul 31, 2024

<https://bit.ly/pwa-navigation-capturing>

AKA - "User Link Capturing"

Reviewer	Status: 🕒 Requested, 🙋 In Review, ✓ Approved	Date	Comments
scheib@chromium.org	✓	Aug 30, 20...	
reillyg@chromium.org	✓	Sep 12, 2024	
robko@chromium.org	✓	Aug 2, 2024	
nasko@chromium.org	🕒		
tsteiner@chromium.org	✓	Sep 12, 2024	
yuchanghu@google.com	✓	Aug 16, 20...	

One-page overview

Summary

Navigation capturing allows an app to 'capture' a user's click on a link to launch their app for that link url. The current implementation of navigation capturing has unacceptable bugs and the feature behavior must be re-evaluated and reimplemented. This will result in about 2 extra SWE months of work required, and pushes the feature back from m121 to likely m130/31.

This feature is being released on Desktop platforms, specifically Windows, Mac, and Linux (ChromeOS already has a shipped implementation that is unacceptable for our use-cases)

The new implementation can be described as 4 things:

1. **User Link Capturing:** Tightly scoped navigation capturing to occur when a user clicks on a link controlled by an app, in a way that is compatible with the web platform, controllable by the user.
2. **Same Browsing Group Fixes:** Keep newly created frames in the same [container](#) (app or browser) if the site opens an `about:blank`.
3. **App Windows from Modified Clicks:** Define what all types aux-click does & when it opens an app window.
4. **Redirection support:** Redirections can become capturable for safe cases.

Platforms

Mac, Windows, Linux.

Sketch/future: reimplement on ChromeOS

Team

pwa-dev@chromium.org

Bug

crbug.com/351775835

Code affected

`browser_navigator.cc`, Navigation throttles, WebAppProvider system, and other minorly related code..

Design Doc

bit.ly/pwa-navigation-handling-dd

Previous design (Internal Only): [go/link-capturing-design-wml](#)

Goals

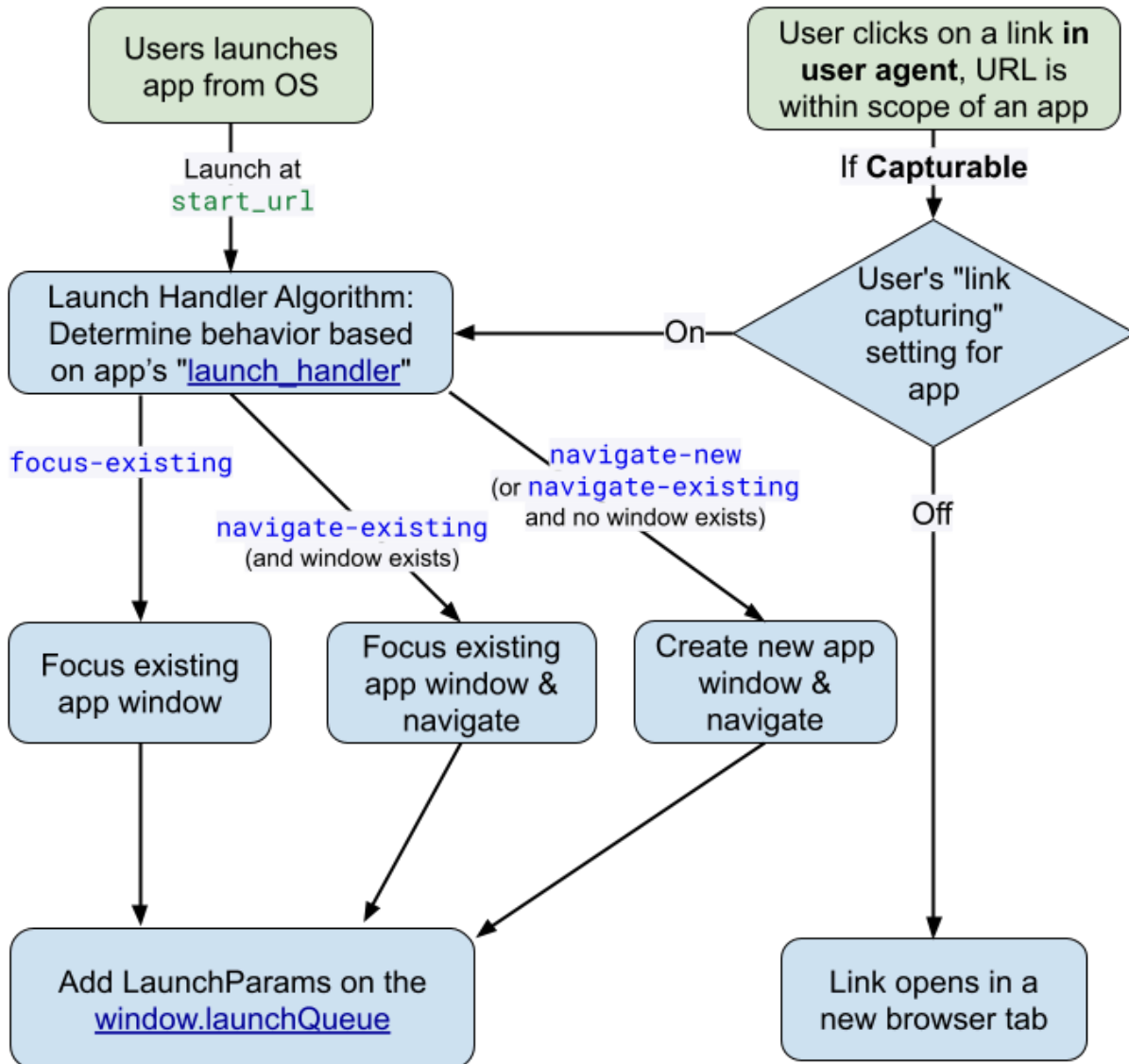
Feature Goal: Allow users to experience sites they have elected to install from link clicks.

"I installed Google Chat, when I click on a link to a Google Chat I want it to open in the app, not in a new tab"

1. **User Link Capturing:** Support navigation capturing to occur when a user clicks on a link controlled by an app, in a way that is compatible with the web platform, controllable by the user.
 - Support `target=_blank` links that rely on opener relationships like (customized) ctrl-click in Gmail.
 - Support `target=_blank` links that use Form post relationships (crbug.com/331246741).
2. **Same Browsing Group Fixes:** Keep newly created frames in the same [container](#) (app or browser) if the site opens an auxiliary browsing context.
3. **App Windows from Modified Clicks:** Define what aux-click behavior should be so that this can open app windows when appropriate.
4. **Redirection support:** Redirection should result in the same behavior as if the final URL was the one that was first navigated to.
5. Stretch / V2
 - POST requests should also be capturable.
 - Link clicks can also be captured from the OS while chrome is the default browser (easy)
 - Link clicks can also be captured from the OS while Chrome is not the default browser (harder, requires registering url patterns in the OS for the app)

Background- User "Link Capturing" & Launch Handling

The following graphic illustrates the basic flowchart for user link capturing & launch handling:



Happening today: when a user launches an app from the OS or from chrome://apps, it "launches" the app at the app's `start_url`, following the above launch handler algorithm.

Sadly, this is currently only implemented in the user agent. Integration with the OS for this feature is a future task below: [Allow link capturing from links in the OS.](#)

Launch Handling

The [launch handling](#) algorithm is used to determine the behavior of loading that url for that app, which the app controls. This can:

- Open a new window at the link url for that app.
- Focus on an existing app window and load the link url there.

- Focus on an existing app window & do nothing.

After, the [LaunchParams](#) is added to the [LaunchQueue](#) on the web contents, for the site to handle as it wishes.

Link Capturing & User Setting

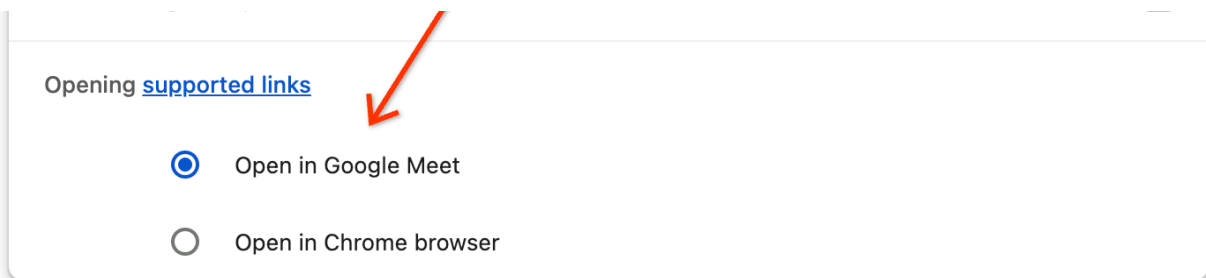
When a user clicks on a link that is determined to be 'capturable' and the user has link capturing turned on for that app, the click is considered 'captured', and its navigation is sent to the launch handling algorithm.

Thus, **capturable** means that the link click action can be captured by an app if:

- The app controls the url (AKA the url is within scope of the app).
- The 'link capturing' user setting on.

Defining when a link is 'capturable' is part of this document's proposal below.

This behavior was controlled by a user setting in the app's settings page like so (launched on ChromeOS today):



The launch on Windows/Mac/Linux had two experiment arms: Default on, and default off. We hope to ship the 'default on' version, where all installed apps capture links by default, and the user can turn it off (we have IPH to help them).

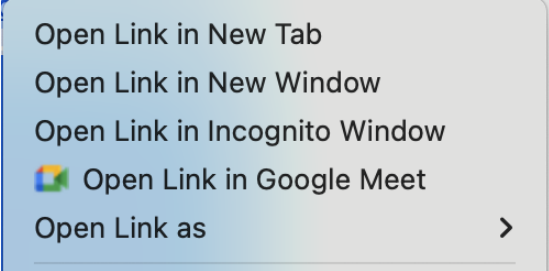

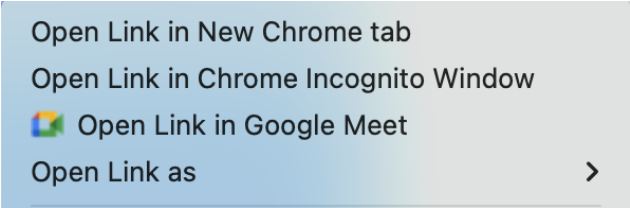

Test with `chrome://flags/#enable-user-link-capturing-pwa`

Links:

- [History](#) below for CrOS launch & documents.
- See also [PWA Link Capturing and URL Handling - The New Taxonomy 2021](#)

Background - User-modified link clicks / Aux clicks

There are many ways users can modify a link click. For example, here is the right-click menu on a link, assuming that link is to an installed app that has link capturing turned on (Google Meet):

From the Chrome browser:	From an app browser
 <p>Open Link in New Tab Open Link in New Window Open Link in Incognito Window  Open Link in Google Meet Open Link as ></p>	 <p>Open Link in New Chrome tab Open Link in Chrome Incognito Window  Open Link in Google Meet Open Link as ></p>

The user can also hold keyboard keys during a link, and these can apply to most of the ways that a site can present a link:

- **Ctrl/Command-Click:** Open the url in a new (background) tab.
- **Shift-Click:** Open the url in a new window.

One of these link clicks is called a **"user modified" link click**.

"User modified" link clicks mostly create a fresh top-level traversable, all links can be clicked in this way, and the [spec](#) has a section defining what this can do. **However**, **`window.open()` can behave differently** - see [Edge Case: window.open\(\) with User-modified clicks](#).

Background - "Same Browsing Group Fixes"

A link click can create different types of frame configurations - it can 'target' a frame on the current document, the current document itself, or a new frame. It can also make a relationship with the new frame, where it receives a javascript reference to the `window` (and the new frame's `window` has an `opener` property) to reference the creator. Those windows / frames (traversables) are part of the same browsing context group.

Basically when a new traversable is being created in the same browsing context group, the new traversable must maintain the same window context (app or browser).

This is the largest area of 'breakage' that the current implementation has, see the [problems with current implementation](#) below.

Background - Terminology

To fully describe how this "same browser group" is happening & the different ways links can occur, the following terminology is necessary.

[Navigable](#) / [Traversable](#) / [Browsing Context](#) ([go/browsing-contexts](#)) - For our purposes, we can generally use these names interchangeably. Each navigable has:

- "window" - the window associated with this.
- "target name" - the target name associated with this.
- "opener" - if the current navigable has an opener, or `null` if not.

Top-level browsing context / traversable - This navigable has its own group, does not have a parent, and `opener` is null. Spec [algorithm](#), and called from [creating](#) a top-level navigable.

Auxiliary Browsing Context - This has a `window.opener`, and is part of a browsing context group with its "opener". This is created by default from `window.open` or from an anchor tag if it specifies `rel="opener"`.

Browsing Context Group / Same Browser Group (spec [link](#) and [navigables](#), see [go/browsing-contexts](#)) - Browsing contexts that can refer to each-other via things like `window.opener` and `window.frames` are part of the **same browsing context group**.

Browsing contexts in the same group are basically tightly-coupled and can reference each other through `window.opener` and (sometimes) `window.frames`. Same-origin browsing contexts in the same group can synchronously script each other, and thus share a process & thread.

Otherwise, they may be distributed across processes, and can communicate asynchronously with `postMessage()`.

Navigation type: [Anchor tag](#)

- `rel="opener"` means the new browsing context will be part of the same browsing context group as this window, and will have the `window.opener` property set.
- **Note: By [default](#), `target="_blank"` links are `rel="noopener"`** (and thus no `window.opener`, and a **top-level browsing context** is created)

JavaScript

```
<a href="www.foo.com/a" target="_blank">create window</a>
<a href="www.foo.com/b" target="_blank" rel="opener">create
window</a>
<a href="www.foo.com/c" target="_self">navigate current
window</a>
```

Navigation type: [Window.open](#)(url, target, windowFeatures)

- Javascript API that opens a window and navigates it to the given url.
- This API synchronously returns a reference to the created window.
- **Note: By default, "window.opener" is set on new windows unless "noopener" or "noreferrer" is specified.** (and thus a new **auxiliary browsing context** is created).

- User-modified clicks only apply selectively when calling this API, as it doesn't take a JS event (but sometimes references the current one). See [this table](#) for how they work.

JavaScript

```
window.open("www.foo.com/a", "_blank")
window.open("www.foo.com/a", "_blank", "noopener")
window.open("www.foo.com/a", "_self")
window.open("www.foo.com/a") // The default target is "_blank"
```

Navigation type: Clients.[openWindow\(url\)](#) - Serviceworker API to open a new client window.

- Always creates a new **top-level browsing context**.

JavaScript

```
clients.openWindow(e.notification.data.url).then((windowClient) =>
(windowClient ? windowClient.focus() : null));
```

Navigation type: "about:blank navigation" - `window.open('').location.href = url`

- This always creates an auxiliary browsing context, as the window reference is given back to the creating page.

[Window.opener](#) - This property is a direct javascript reference to the window that opened this current window. This being set means that the current window and the opener are in the same browsing context group.

Rel "noopener" and "noopener"

- **"noopener"** means that `window.opener` isn't populated, and it will create a new browsing context group.
- **"noopener"** automatically implies `noopener`, and also means that referrer information isn't included in the network request [headers](#). This option is less common, as often the 'referrer' information is required for SEO.

Target ([MDN](#), [spec](#)) - For both anchor links and `window.open`, this specifies where the link will load. The important target here is `_blank`, which is the only target that is defined to always create a new top-level browsing context.

Container (invented term here). "App A", "browser tab", "App A Tab" - This is the style of window a browsing context lives in. This can be a browser tab, an app window, or an app tab in 'tabbed' mode.

Choosing a navigable for a navigation (link click / `window.open`): - [spec algo](#) - Given an optional frame name, this algorithm determines if:

- An existing browsing context is used.
- A new top-level browsing context will be created.
- A new auxiliary browsing context will be created with optional frame name.

Background - Redirection

Redirects make determining what context to open a link in more tricky, since the origin (and app) associated with the link that was clicked might not be the same as that of the URL that ultimately gets loaded.

Redirects can be split into server side and client side redirects. Server side redirects are triggered by the server or service worker returning a 30x status code, while client side redirects are implemented in javascript or using a `<meta>` tag after a page has loaded. As such client side redirects are not “true” redirects, but Chrome does have [code](#) to identify certain client side navigations as redirects.

Background - Putting it all together & common behavior

Important conclusions & behaviors given the above terminology.

- When a user does a modified click on a link, it always opens in a new top-level browsing context.
- When a user clicks on an anchor tag with `opener` or `window.open` without `noopener`, this creates an auxiliary browsing context.
 - This means that the new browsing context is part of a shared browsing context group with the parent.
- The cases that create a new top-level browsing context (NOT an auxiliary browsing context):
 - An anchor tag with `target="_blank"` without `rel="opener"`.
 - A call to `window.open(url, "_blank", "noopener")`
 - All calls to `clients.openWindow` from a serviceworker.
- **Browsing contexts that are in the same group often affect each other.** Often closing a parent browsing context will close its auxiliary contexts (via dev logic), as the auxiliary contexts depend on the parent. (see how gmail's [custom ctrl-click window](#) works).

Background - Problems with current implementation

The current implementation intercepts links using a [navigation throttle](#) ([Design Doc](#) (Googler only)) in the browser, which occurs after a WebContents is already created & added to a browser window. If a navigation is 'capturable', it cancels the navigation & launches the applicable app at the url.

This is bad because:

1. All navigation data & parameters are blown away (like window.opener, originator, etc), breaking flows & some security things.
 - o Internal bug: issuetracker.google.com/318422881
2. Browser flashes for app-to-app capture
 - o When capturing between two apps (e.g. click on a youtube link from gmail app to launch youtube), the browser is focused briefly as the interception happens after a new WebContents is created & navigation begins.
3. Further bugs:
 - o issuetracker.google.com/319142353
 - o crbug.com/338216059: User link capturing breaking Chrome Remote Desktop

Gmail example: Gmail overrides ctrl-click on emails, opening a new window using `window.open`, and that window loads the email contents using `window.opener`. When the gmail parent window closes, it iterates over all created frames like this and closes them. (If it doesn't, the `window.opener` becomes null in that frame.) Internal bug: <http://issuetracker.google.com/318422881>

Additionally the behavior exhibited by the CrOS implementation seems incorrect for Web Platform links:

- Link navigations that specify a frame (`_self` or a specific frame name) are capturable, even if a frame with that name already exists.
- Other bugs noticed in (Googler only) [Link Capturing use-cases for URL Navigations](#), like `window.open` from inside an app never launches an app window.

Design

Non User-Modified Clicks Changes

Assume that:

- A and B are apps and urls for respective apps
- C is not an installed app.

- The link click isn't modified by the user.

Current URL, Container	URL for all types of link navigations	Browsing context type created	Resulting behavior (bold = new behavior)
A, App A	A	new top-level browsing context	Capturable by App A
		new auxiliary browsing context	new App A container
		existing browsing context	navigate
	B	new top-level browsing context	Capturable by App B
		new auxiliary browsing context	new App A container, with 'custom tab' banner (standard)
		existing browsing context	navigate with ' custom tab ' banner (existing behavior)
	C	new top-level browsing context	Browser tab
		new auxiliary browsing context	new App A container, with 'custom tab' banner (standard)
		existing browsing context	navigate with ' custom tab ' banner (existing behavior)
A, browser tab	A	new top-level browsing context	Capturable by App A
		new auxiliary browsing context	Browser tab (or window)
		existing browsing context	navigate
	B	new top-level browsing context	Capturable by App B
		new auxiliary browsing context	Browser tab (or window)
		existing browsing context	navigate

Should we be enqueueing launch params for auxiliary context creations?

Probably not, since there can be an out of scope navigation happening in the same app context, and it doesn't make sense to queue those params.

User-Modified Clicks Changes

Here is a table of the behavior for "user modified" link clicks.

Summary / precondition: **To keep this simple, this 'new behavior' should only happen if the navigating-to app has the 'link capturing' user setting turned on.** Otherwise the behavior is unchanged from the current behavior.

Exception: window.open() with auxiliary browsing contexts will work independent of the link capturing user setting, see this [section](#) for more information.

Current URL, Container	navigating-to URL	Custom Link User Action	Proposed behavior (bold = new behavior)
A, App A	A (installed app)	shift-click	new App A container
		middle-click / metakey-click	new App A container *
		context-menu: new tab	new browser tab
		context-menu: new window	N/A
		context-menu: open link in A	new App A Container
	B (installed app)	shift-click	new App B container
		middle-click / metakey-click	new browser tab
		context-menu: new tab	new browser tab
		context-menu: new window	N/A
		context-menu: open link in B	new App B Container
	C (not an app)	shift-click	new browser window
		middle-click / metakey-click	new browser tab
		context-menu: new tab	new browser tab
		context-menu: new window	N/A
		context-menu: open link in	N/A

A, browser tab	A (installed app)	shift-click	new browser window
		middle-click / metakey-click	new browser tab
		context-menu: new tab	new browser tab
		context-menu: new window	new browser window
		context-menu: open link in A	new App A Container
	B (installed app)	shift-click	new browser window
		middle-click / metakey-click	new browser tab
		context-menu: new tab	new browser tab
		context-menu: new window	new browser window
		context-menu: open link in B	new App B Container
	C (not an app)	shift-click	new browser window
		middle-click / metakey-click	new browser tab
		context-menu: new tab	new browser tab
		context-menu: new window	new browser window
		context-menu: open link in	N/A

*** if app A is 'tabbed' display mode, this can open a new tab in the current window rather than a entirely new window**

When user does a modified link click:

- No changes to the behavior of the context menu clicks.
- Otherwise, if coming from an app container A
 - If the url is within scope of A, create a new app A container for the url.
 - If the url is within scope of another app B AND the modification was to open a new window, create a new app B container for the url.

When creating a new app container for a modified click:

- Populate the `launchQueue` with `LaunchParams` for the url, mimicking a 'new-client' launch behavior.

Reasoning:

- This respects the user's choice to open the link in a new container/context.
- This respects the user's current container (browser or app)
- This only opens in an app container if the user is already in an app container

- Populating the `launchQueue` helps ensure behavior is consistent with a launch of that app at the given url.

Edge Case: `window.open()` with User-modified clicks

`window.open()` is executed from javascript, unlike being a native click behavior on the `anchor` element. Thus it doesn't inherently take a click event, and so it might be expected that it doesn't change its behavior for [user-modified link clicks / aux clicks](#). That is not always the case.

When the `window.open` is executed in a click event listener, and the click was a user-modified click:

1. **If the API call targets an existing frame, behavior matches non-user-modified clicks.**
 - a. For example if the `target` is `"_self"`, `"existingFrameInPage"`, or `"alreadyOpenedFrame"`.
2. **If the API call would create a new browsing context, then the user-modified state DOES have an effect, but only in the way the frame is presented.** E.g. new background tab, new foreground tab, new window.
 - a. This occurs when the `target` is `"_blank"` or `"uncreatedFrameName"` or empty.
 - b. Importantly - this will still respect the `opener` state - it can open a new top-level or a new auxiliary browsing context.

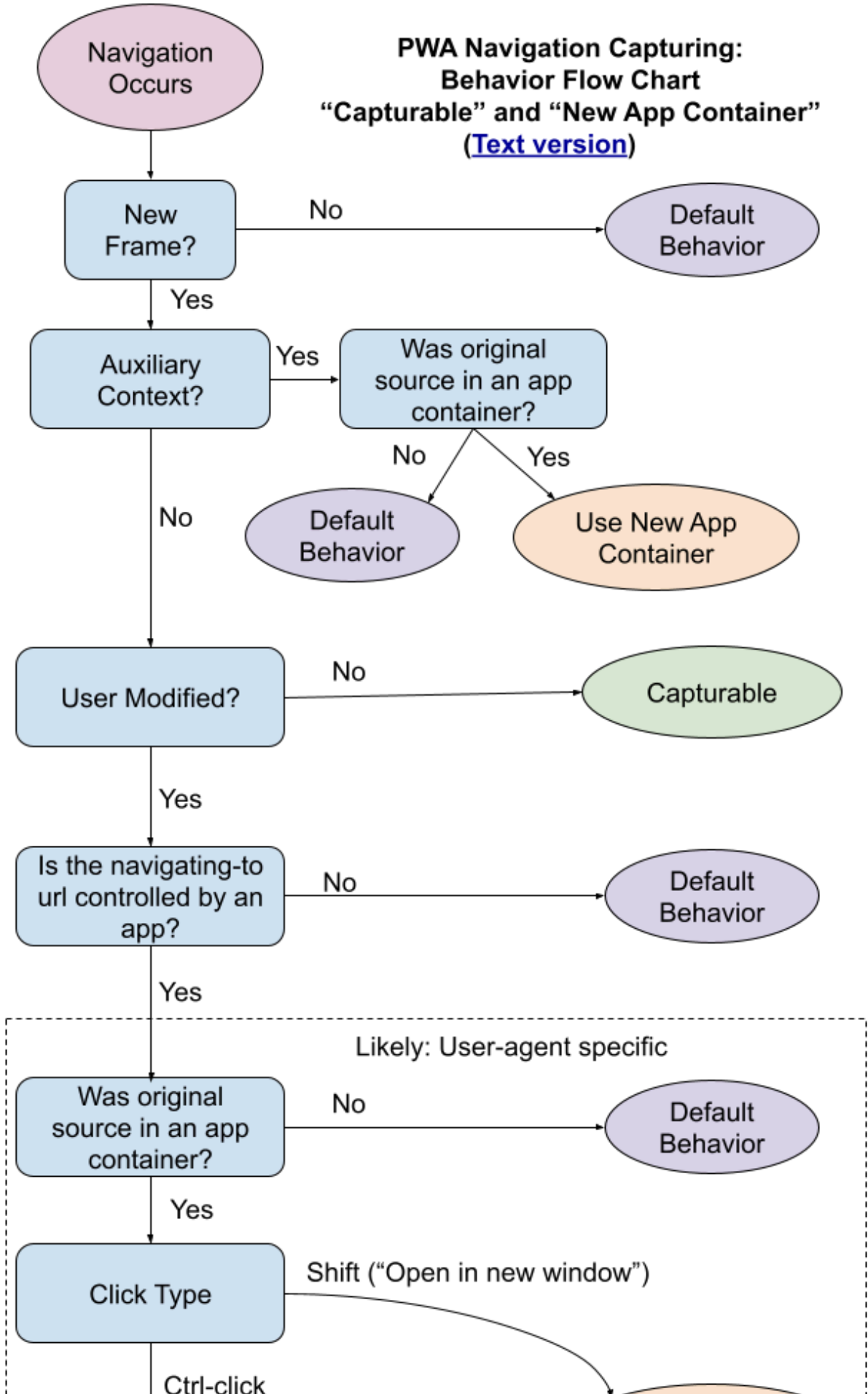
Thus, the proposed behavior for handling `window.open()` + user-modified click is:

Chosen Frame	Browsing Context	Proposed Behavior
Existing	Auxiliary	Current Behavior (No change)
	Top-Level	Current Behavior (No change)
New	Auxiliary	Use proposed behavior for auxiliary browsing context for non-user-modified clicks (but keep modifier for the new frame being created)
	Top-Level	Use proposed behavior for user modified link clicks

Putting it together: Behavior Flow Chart for "Capturable" and "New App Container"

Below is the resulting behavior flowchart for this feature.

**PWA Navigation Capturing:
Behavior Flow Chart
“Capturable” and “New App Container”
(Text version)**



Text version: [Text version of the Final Nav Capturing Flowchart \(non-redirect\)](#)

Redirection Handling Section

Ideal Result: The final behavior of a redirect will be the same as if there was never a redirect, and the final post-redirect URL was the original URL being navigated to.

This behavior is possible, although in the process another window or tab might flash briefly.

Product behaviors

- If the first URL is captured & focus-existing, the chain doesn't occur at all.
- Otherwise, the final state will match as if the final url is the first url navigated to.
- Edge cases
 - Sometimes a window flashes on-screen before navigation completes.
 - Sometimes we will fetch twice.

At the point in time when we determine how to handle a specific navigation above we don't know yet if later redirects might happen. As such in the case of redirects, the "regular" non-redirect behavior will happen first, and afterwards when we reach the end of the redirect chain one of three different things can happen:

1. Nothing special, complete the navigation in the tab/container picked originally.
2. We'll reparent the tab being navigated into a new or existing app or browser window
3. We'll abort the original navigation and trigger a launch event and focus an existing app window. In case of a server-side redirect we'll also close the tab being navigated originally if this was the first navigation in that tab.
4. We'll abort the original navigation and trigger a new navigation in an existing app window, preserving as much as possible of the original navigation. If the original navigation was the first navigation in the tab being navigated, we'll also close the tab that was being navigated.
 - a. See [Proposal: Redirection with Capturable to "navigate-existing" apps will cancel navigation, start a new navigation in an existing app window](#) below, and [Alternative: Other options for redirection + navigate existing](#)
 - b. Note: This is different from the "navigate-existing" behavior for non-redirect navigation. In the non-redirect case we can immediately pick the tab to navigate, starting the navigation there; but with redirects unfortunately we have to restart the navigation in the new location after detecting the redirect.

Proposal: Redirection with Capturable to "navigate-existing" apps will cancel navigation, start a new navigation in an existing app window

This most closely matches the intended behavior. Chrome already has methods to create an `OpenURLParams` instance from a `NavigationHandle` forwarding all the relevant information from the original navigation to a new (`WebContents`) navigation, so this also would be fairly simple to implement. There is a slight risk that some information about the navigation is lost in this conversion.

Note that while similar, this is explicitly not the same as the current (broken) navigation capturing implementation has. We're not triggering an "app launch" with the target URL, we're directly navigating the target web contents while maintaining the full metadata of the original request, including things like the redirect chain.

How these changes can be reflected in the HTML specification.

Change 1 (Capturable): When creating a [new top-level traversable](#)

- If
 - The URL being navigated to is within scope of an app AppBeingNavigatedTo.
 - The `opener` is `null` (AKA it's NOT an auxiliary browsing context).
- This is capturable, and the behavior now depends according to the [launch handler](#) of App "navigating to"
 - "navigate existing" will instead cause this algorithm to return an existing traversable, if it exists for the app "navigating to".
 - "navigate new" and "auto" will create a new traversable as standard, put it into an App "navigating to" window.
 - "focus-existing" will abort this whole process and simply focus an existing App "navigating to" window.

Example: An anchor link with a target of "`_blank`" is left-clicked by the user, and the url is within the scope of an installed app.

Change 2 (Same Browsing Context Group = Same Container): When creating a [new top-level traversable](#)

- If
 - The `opener` is NOT `null`. (AKA it's an auxiliary browsing context).
 - If the current container is an app AppBeingNavigatedFrom.
- Create a new app AppBeingNavigatedFrom window for the new traversable.
 - (This means that any url provided will open into an App A window, even if it is out of scope.)

Example: [Gmail example: Gmail overrides ctrl-click on emails, opening...](#)

Change 3 (User Modified Browsing Context): When creating a [fresh top-level traversable](#) (AKA from a "user modified" link click):

- Same-app case:

- If
 - The current container is an app.
 - The URL being navigated to is within scope of the same app
 - The modifier is a keyboard key (shift or ctrl/command).
- Then
 - Create a new app window for the new top-level traversable.
- Other-app case:
 - If
 - The current container is an app.
 - The URL being navigated to is within scope of another app AppBeingNavigatedTo.
 - The modifier is a keyboard key that results in a 'new window' (shift key)
 - Then
 - Create a new app window for the new top-level traversable.

Examples:

- A user shift-clicks a link to foo.com/app, which is in scope of the Foo app.
 - If they click on this link in a browser window, it will open a new browser window
 - If they click on this link in an app window, it'll open the Foo app.
- A user middle-clicks on a link to foo.com/notapp, which is not in scope of an app, it will always open in a browser tab.

"Same Browsing Context Group = Same Container" Explained: Always open the url in app A window.

Using the [Gmail example: Gmail overrides ctrl-click on emails, opening...](#)

Non-controversial: **Links that create auxiliary browsing contexts from app A to app A will always open in another app A window.** This seems correct - this is how the gmail ctrl-click above will stay in the gmail app.

Potentially surprising: **Links that create auxiliary browsing contexts from app A to any URL will always open in another app A container.**

Why? Browsing contexts that are in the same group can affect each-other. For example, closing gmail will close all of the 'preview' windows that are created, as the developer closes that window as it depend on the parent.

- If we allowed a frame with `window.opener` to be captured into an app while the parent stayed in the browser, that would mean that if you opened an email preview window in Gmail-in-a-browser-tab, then that will open in the gmail app. This is visually separate and can be command-tab switched to as a separate application. Closing Gmail in the tab would close that app window - this is weird!

- Same if Gmail was in an app context & the preview opened in a browser tab. Very weird to have browser tabs close if the Gmail app is closed.

This might not always be expected by developers if they are incorrectly not using `noopener` for `window.open()` external links. However this correctly reflects the tight coupling of these browsing context. If the dev doesn't want external links to open in their app window, then they need to update the links to not contain an 'opener', which is a security risk for them.

Note: `noopener` is implied by default for `<a target="blank_" ...` links, so this should only affect `window.open()` calls that don't specify `"rel=noopener"` in the 3rd (optional) argument. So the impact will likely be small.

Edge Case on Mac: Double-clicking on application, when application is already open.

TODO(mek@) - propose something here, circle back with Rob or Yuchang on expectation.

Options

- Focus the window (which sends an activation event / visibility event) but don't do anything else.
- Same, but add launchParams as we're doing the `focus-existing` behavior.
- Same, but fully follow the site's launch handler `clientMode` - sometimes we'll create a new window, sometimes we'll navigate existing, sometimes we'll just focus on existing.

Edge Case: Target name specified, but opener is null (or noopener)

When a frame name is specified, the frame doesn't exist on the page (and hasn't been created before with an `opener`), and `opener` is null, then the link effectively acts as `target="_blank"`, but the window has a frame name that doesn't seem to do anything.

- Modified case: If a frame name was specified in the past with a non-null `opener`, then the above behavior would navigate in the previously opened frame.

Proposed behavior: Only check `opener` on newly created contexts to determine if it is an auxiliary context. This means all of the clicks in the example above are considered non-auxiliary. But in the modified case the first would be auxiliary and subsequent clicks don't create a new browsing context, so none of our code would trigger.

"Popup" or "Normal" window?

We want to respect existing behavior for auxiliary contexts for when a popup window is created. So if a window would have been a popup window, then it should continue to be so.

Code locations

See <https://bit.ly/pwa-navigation-handling-dd>

Decision - 'User modified' link clicks that open app windows queue a LaunchParams

The middle-click case could also put an item on the launch queue, to better model how launching will now always do this when launching in an app context. This could be captured with a launch handler override of "new-client" and behave as usual, no suppression of launch queue events.

Con:

- This is a bit weird & might be unexpected.

Alternative: Other options for redirection + `navigate_existing`

Don't support "navigate-existing", re-parent into a new app window instead

This doesn't match what the site asked for, but does not interrupt the in-progress navigation, and still results at least in the redirect ending up in the correct app.

Don't capture redirects to "navigate-existing" apps at all

This kind of "punished" sites for selecting "navigate-existing". If they didn't use that launch behavior navigation would have been captured but they aren't because of it. While easy to implement this seems strictly worse than capturing into a new window.

Swap the existing tab/web contents with the one being navigated

There used to be code in Chrome to swap out a WebContents with another for features like <portal>. That code was largely removed though as it proved to be too much of a maintenance burden. As such re-introducing that for this feature seems undesirable.

Alternative - Middle-clicks should be capturable

For product change case 1), this could just enter the capturable pipeline.

Pros:

- Gives devs control over middle-click case

Cons:

- Opens room for developer footgun to not open new context when user wants to

- Devs can already preventDefault() this and customize it if they want to.

Alternative - Middle-clicks should never open in an app

For product change case 1), this could always open in a browser.

Pros:

- Very predictable

Cons:

- Users have said they want links to X to capture in X, so they likely want it to open in apps sometimes.

Alternative - Middle-clicks should always open in an app window.

For product change case 1), this could always open in a browser.

Pros:

- Very predictable
- Users have said they want links to X to capture in X, so they likely want it to open in apps sometimes.

Cons:

- Hard to open a tab if you turn on user link capturing.

Alternative - Add new target types or html attributes for link capturing

Possible product change, too much work for MVP.

If the developer wants the user to always open in an app, they can either do that by a new html attribute `link-capture=` or a new target type `target=link-capture``.

As long as the flag is enabled, the link would always open in a new top level browsing context in the app if installed, otherwise it will always open in a new tab.

Possibly requires changes to spec and a bunch of other places to drive developer acceptance and usage.

Metrics

Success metrics

You should list what metrics you will be tracking to measure the success of your feature or change. This could be a mix of existing and new metrics. If they are new metrics, explain how they will work. If you aim to improve performance with your feature or change, you should measure your impact on one of the [speed launch metrics](#).

Regression metrics

You should define what metrics you will be tracking to look for potential regressions associated with your feature or change. This could be a mix of existing and new metrics. The [speed launch metrics](#) are good candidates for use as performance regression metrics. If you're using new metrics, explain how they will work.

Experiments

If you are using [Finch](#) to run experiments (see [go/newChromeFeature](#) for advice), describe what experiments you intend to run and what you are looking for in the results. It's important that you know in advance what the acceptance criteria are. List the experiment names so people can look them up (links to the [dashboard](#) are even better).

Rollout plan

If you're just checking in the code and doing a dev-beta-stable progression, just write "Waterfall" here. If you're doing a standard experiment-controlled rollout, write that with the experiment name. If you're not doing the standard rollout, describe what you're doing and why it needs to be different.

If there are external deadlines, call them out (if you don't want these to be public, use the [internal template](#)). Otherwise, releases should be quality driven and you shouldn't be targeting a milestone.

Core principle considerations

Everything we do should be aligned with and consider [Chrome's core principles](#). If there are any specific stability concerns, be sure to address them with appropriate experiments.

Speed

This implementation greatly increases the speed of captured navigation as

- This no longer destroys a `WebContents` that is already loading the page to recreate it using the `WebAppLaunchProcess`.
- This is compatible with prerendering.

Simplicity

This integrates tightly with already existing navigation logic, reusing already supported features like choosing an existing tab to load a navigation.

Security

The main security concern is ensuring that the navigation initiator properties are preserved, which includes all of the properties in `NavigateParams` that have been constructed during a navigation. Other properties that are not included in `NavigateParams` but could be preserved include the following (not comprehensive):

- [ReferredPolicies](#).
- [WebSandboxFlags](#)
- [PolicyContainerPolicies](#)
- [Ip address space](#)

Doing this was essentially the whole premise of this design - the current implementation does not do this at all.

Non-risks:

- Spoofing: There is no way to pretend to be another app here - the most a malicious website can do is present a link to a url controlled by an installed app, and have it possibly open that installed app. This is the same behavior as if that link was loaded in a browser tab for that app.
-

Privacy considerations

N/A

Testing plan

Browsertests should be able to handle everything here. See <https://bit.ly/pwa-navigation-handling-dd>.

Followup work

Allow link capturing from links in the OS

Possibly technical complexity about PWAs installed by Chrome when the default browser is Safari. Not sure how to pass that information in just a link.

There is a lot of interest here though, especially from third-party users like Zoom and Salesforce.

Also brought up in the public tracking bug:

<https://issues.chromium.org/issues/40255471#comment61>

Need to prototype if this can be done, but this might require some form of OS integration once a PWA is installed.

Salesforce Request: crbug.com/330282442

Browsing context group switches

Documents can set the Cross-Origin-Opener-Policy header to `same-origin` or `same-origin-allow-popups` to prevent them from being included in a browsing context group with cross-origin content. When a cross-origin document is loaded into a top-level frame, and either the parent or popup frame have this header, the popup is [kicked out](#) of its opener's browsing context group and loaded into a new group. Its `window.opener` will be null.

Navigations that get kicked out of their opener's browsing context group due to headers will not have launch params enqueued. Maybe they should? This has the potential to impact IWAs.

Popup window behavior

This table assumes a page is calling `window.open(popupUrl, '_blank', 'popup')`:

Current URL, Container	Current COOP Value	Popup URL	Popup COOP Value	Popup Container	Popup <code>window.opener</code>
A, App A	*	A	*		App A
A, App A	unsafe-none	B	unsafe-none		App A
A, App A	unsafe-none	B	same-origin[-allow-popups]		null
A, App A	same-origin	B	*		null

A, App A	same-origin-allow-popups	B	unsafe-none		App A
A, App A	same-origin-allow-popups	B	same-origin[-allow-popups]		null

Service Worker Launch handling behavior

SW launch handling cannot support window.open, as that is a synchronous JS API that immediately returns a window reference, and SWs are inherently asynchronous.

Open question - Navigate self OR capture behavior

TODO:

- Add use-case for a site that wants to create a link w/ the behavior "link capture this if the app is installed w/ link capturing turned on, otherwise navigate the current frame". e.g. Google. This might require a new target attribute. This can be remedied by link capturing + middle click behavior outlined below.

Open question - ChromeOS Behavior

The desired behavior from ChromeOS is unclear and requires confirmation from their product team. ChromeOS will need to provide resources to rectify their implementation with our final decision here for the Web Platform behavior on Windows/Mac/Linux.

Complications that may lead to a difference in behavior:

- Android apps expect to be launched if a link url is within that apps scope, no matter what (even if it's navigating self, etc).
- Others?

Appendix

Alternative User-Modified Clicks Changes, allowing more app-to-same-app -> app container

This adds extra configuration allowing app-to-same-app creation to work whether or not link capturing is turned on.

Current URL, Container	navigating-to URL	Custom Link User Action	'Link capturing' User setting	Proposed behavior (bold = new behavior)
A, App A	A (installed app)	shift-click	on	new App A container

			off	new browser window
		middle-click / metakey-click	on	new App A container *
			off	new browser tab
		context-menu: new tab		new browser tab
		context-menu: new window		N/A
	context-menu: open link in A		new App A Container	
	B (installed app)	shift-click	on	new App B container
			off	new browser window
		middle-click / metakey-click	on	new browser tab
			off	new browser tab
		context-menu: new tab		new browser tab
		context-menu: new window		N/A
	context-menu: open link in B		new App B Container	
	C (not an app)	shift-click		new browser window
		middle-click / metakey-click		new browser tab
context-menu: new tab			new browser tab	
context-menu: new window			N/A	
context-menu: open link in			N/A	
A, browser tab	A (installed app)	shift-click		new browser window
		middle-click / metakey-click		new browser tab
		context-menu: new tab		new browser tab
		context-menu: new window		new browser window
		context-menu: open link in A		new App A Container
	B (installed app)	shift-click		new browser window
		middle-click / metakey-click		new browser tab
		context-menu: new tab		new browser tab

		context-menu: new window		new browser window
		context-menu: open link in B		new App B Container
	C (not an app)	shift-click		new browser window
		middle-click / metakey-click		new browser tab
		context-menu: new tab		new browser tab
		context-menu: new window		new browser window
		context-menu: open link in		N/A

History

The CrOS team originally launched this feature in 2016 ([bug](#)), enabling Android-style intent linking to installed ARC apps, which doesn't necessarily align with web platform expectations for how link clicks work. In 2021, this implementation was expanded to also handle links to web apps. The core link handling logic is old and mostly predates the current CrOS team members owning this feature.

The Web Platform team picked this up in mid-2023 to launch on Windows, Mac, and Linux, and directly modeled their implementation on the already-shipped CrOS implementation.

The CrOS implementation has no documentation for "what are applicable web platform link clicks", and it seems like this exploration was never done or at least never written down. There are various 'bugs' fixed for the CrOS link capturing implementation into 2024, with unfixed outstanding bugs of the browser 'blinking' on app-to-app capture, and Gmail ctrl-click link capturing being broken.

CCT / 'custom tab' banner

See the banner with the 'x' button below the title bar. This automatically shows for any url that is showing in an app that is not within scope of the app.

