Fix PWA media system prompts for macOS

Dev Design Spec

Authors: Lia Hiscock, Stanley Hon

Spec Status: Ready for Review

Quick Links
PM spec: Support for SystemMediaControls per dPWA - Google Docs
Windows dev design spec: Chromium Dev Design Spec – Fix Media System Prompts on Win11.docx - Google Docs
How to test: How to test CL SystemMediaControls per web app - Google Docs
Prototype: [WIP][dPWA][Mac][SystemMediaControls] bridged app shim design v2 (5493397) · Gerrit Code Review (googlesource.com)

Reviewers

Name	Required	LGTM/NLGTM
(MSFT) Hoch Hochkeppel	Yes	LGTM – 5/2/2024
steimel	Yes	LGTM - 5/13/2024
mek	Yes	LGTM - 5/13/2024
dmurph	No	

1. Feature Overview

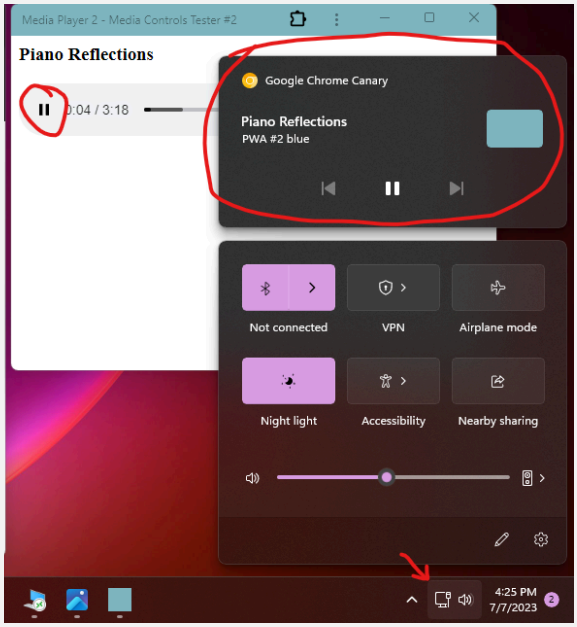
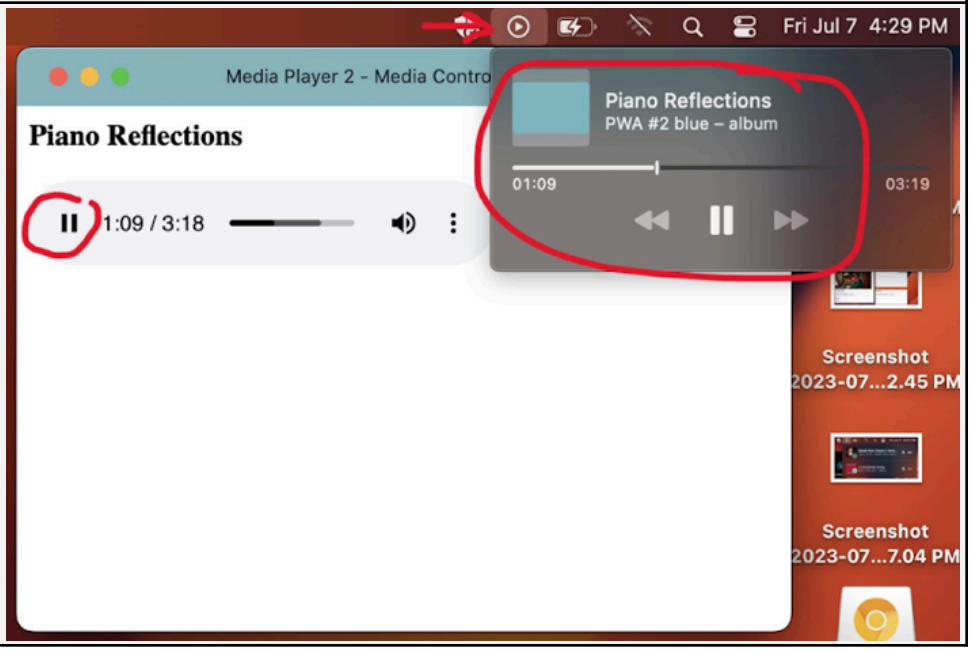
1.1. Motivations

This spec covers the macOS portion of System Media Controls. The motivations for the feature are the same as in the windows spec. This spec focuses on the implementation detail of implementing the feature on macOS.

To summarize in one sentence:

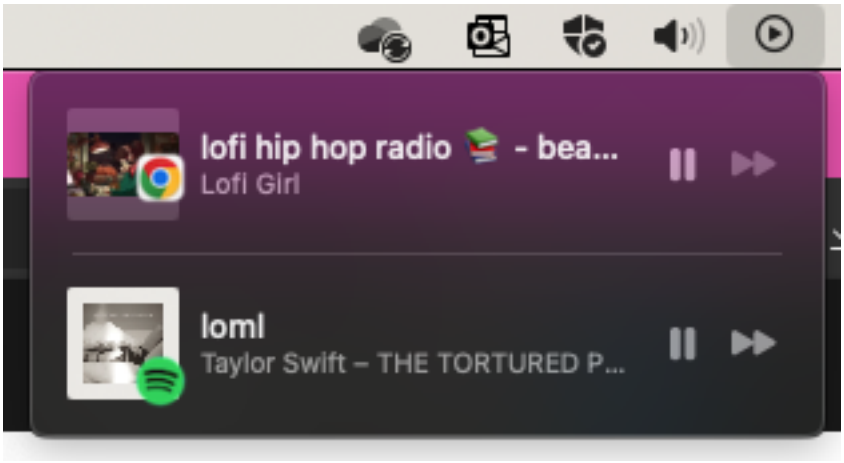
Allow each PWA playing media to have its own set of controls in the macOS interface.

1.2. Background

In Windows, the UI pops up when the "control center" is brought up.	On macOS, you click the "now playing" icon to show this UI.
	

1.3. Goals

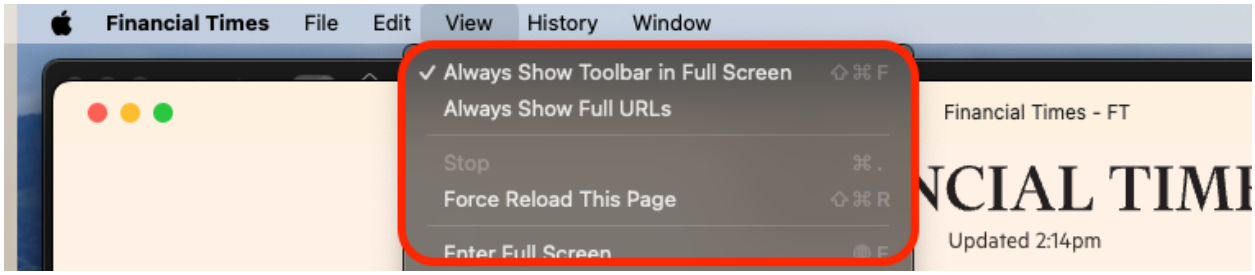
On macOS, the now playing center should have a separate entry for each PWA playing audio, plus one for the browser.

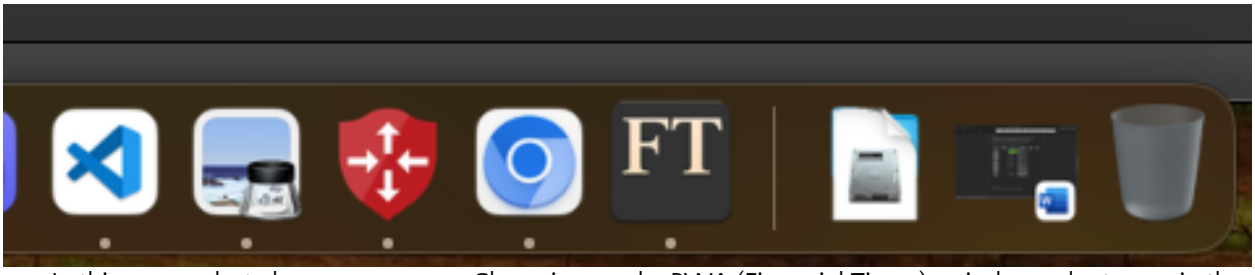


We will show the PWA icon in media controls box, not the browser.

1.4. The App Shim

In Chromium, PWAs utilize a concept known as an App Shim. Its purpose is to present PWAs - which are just essentially browser windows - as a separate "app" on macOS. It also handles a lot of messages to and from the Browser process, such as using macOS menus associated with Financial Times shown below.





In this screenshot above we can see Chromium and a PWA (Financial Times) as independent apps in the macOS dock. The App Shim enables this. It is an independent process to the browser process. The App Shim communicates with the browser over various mojo connections.

1.5. macOS APIs

There are 2 "objects" used to read/write to macOS. Both of them are essentially singleton per-process, meaning you must interact with them from within each app shim process in order to have independent system media controls for each PWA.

1.5.1. MPNowPlayingInfoCenter

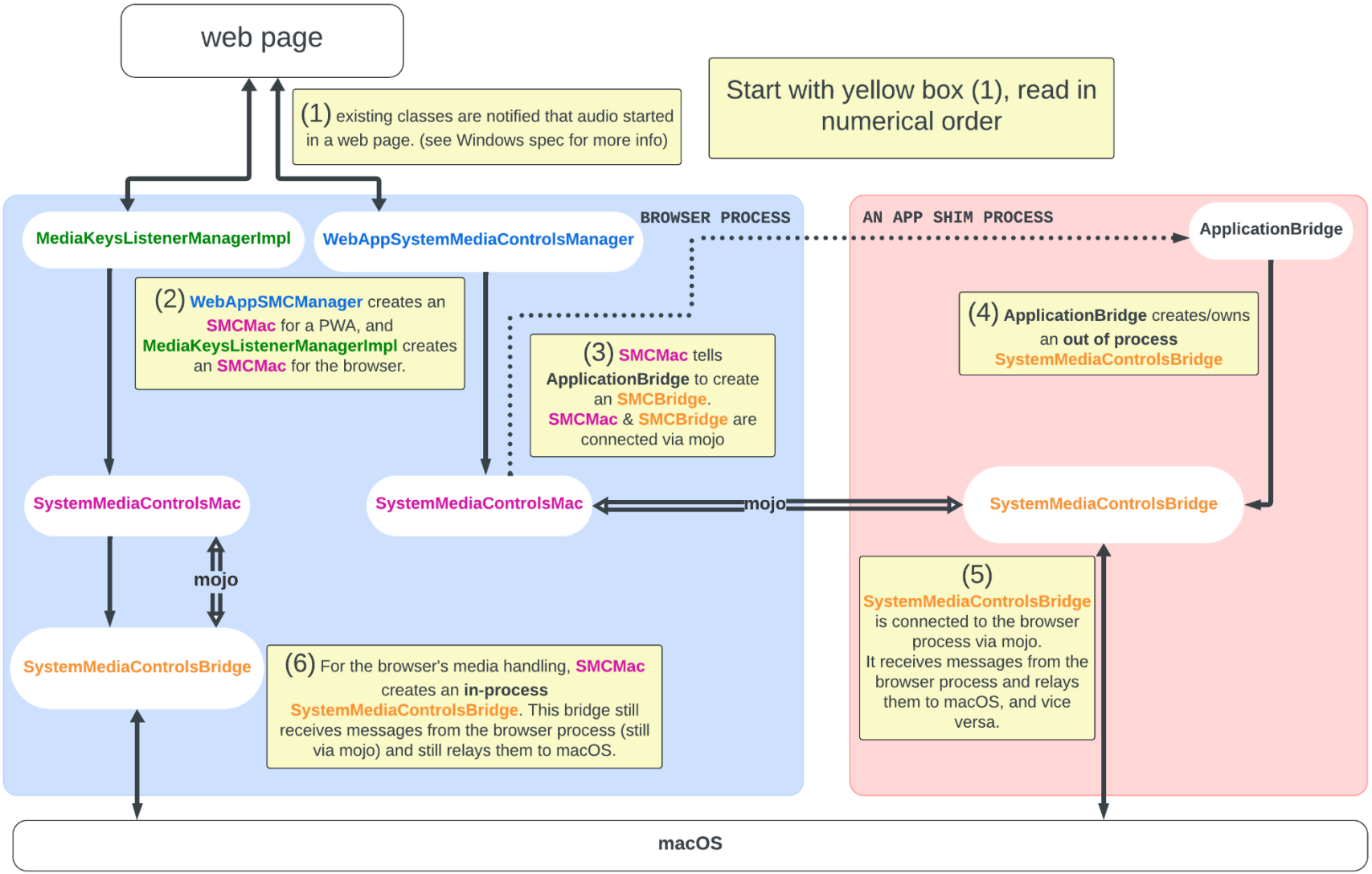
Lets you specify metadata about your media that macOS will display. This includes things such as artist name, album, track title, etc.

1.5.2. MPRemoteCommandCenter

Lets you register handlers so your app can respond to events received from macOS, like plays/pauses.

2. Design

This is an overview of how we will leverage the in-process/out-of-process bridge design pattern with a new **SystemMediaControlsBridge**.



2.1. SystemMediaControlsManager

This section is relevant to step (2) in the design diagram above.

WebAppSystemMediaControlsManager, added during the Windows implementation, creates and manages **one** SystemMediaControlsMac **for each** unique PWA playing audio. (Whereas MediaKeysListenerManagerImpl creates/manages a single SystemMediaControlsMac for all browser tabs/windows).

2.2.SystemMediaControls::Create API changes

This section is relevant to steps (2) and (3) in the design diagram above.

SystemMediaControls(Win/Mac/Linux) objects are created via the platform agnostic Create function. After our Windows implementation, Create looked as follows –

```
static std::unique_ptr<SystemMediaControls> Create(
    const std::string& product_name,                    // used on Linux
    int window);                                        // used on Windows (HWND handle)
```

However, SystemMediaControlsMac needs to access an AppShim's ApplicationBridge to make the out-of-process SystemMediaControlsBridge. We get the ApplicationBridge like this –

```
remote_cocoa::ApplicationHost* application_host =
 remote_cocoa::ApplicationHost::GetForNativeView(web_contents->GetNativeView());

raw_ptr<remote_cocoa::mojom::Application> application_bridge_ =
      application_host->GetApplication();
```

We can't pass the WebContents to Create without causing a circular dependency between //components/SystemMediaControls and content/browser, so we have to pass ApplicationHost, which only builds on Mac. This means we need a Mac-only Create function –

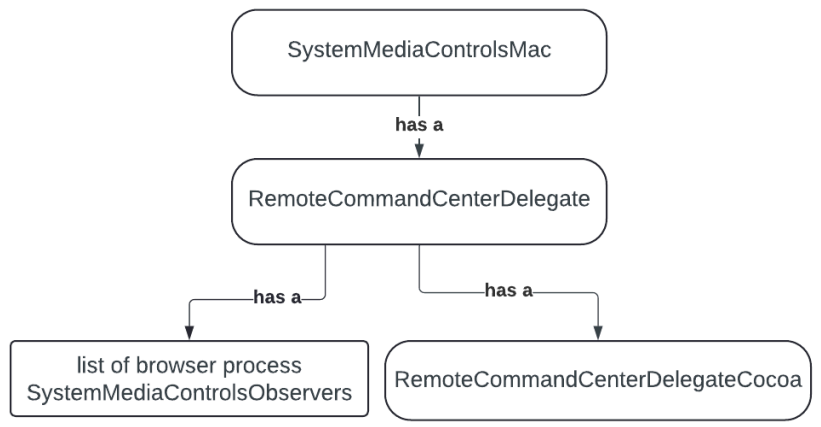
```
#if BUILDFLAG(IS_MAC)
// static
std::unique_ptr<SystemMediaControls> SystemMediaControls::Create(
    remote_cocoa::ApplicationHost* application_host) {
    // |application_host| will be null for the browser
    return std::make_unique<internal::SystemMediaControlsMac>(application_host);
}
#endif
```

2.3. //components/RemoteCommandCenterDelegate (RCCD) changes

This section is relevant to steps (5) and (6) in the design diagram above.

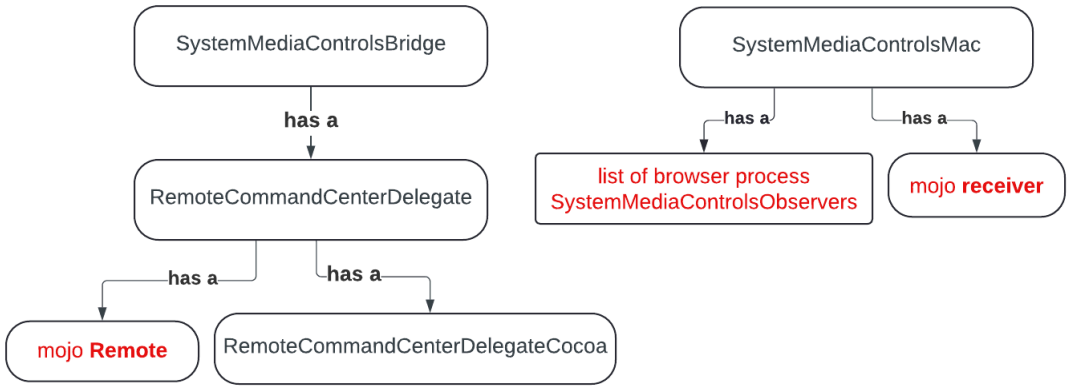
RCCD is the component that interfaces with the macOS APIs, specifically the MPRemoteCommandCenter which allows us to listen for events from macOS and update the web contents accordingly. The following changes prevent duplication of a significant amount of code.

Before – RCCDCocoa directly received the events from macOS, passed them up to RCCD, which then forwarded the event to all its observers (for us, this is namely MediaKeysListenerManagerImpl).



The problem – MPRemoteCommandCenter is essentially a singleton per-process, so RCCD needs to run in the app shim process, but the list of SystemMediaControlsObservers must stay in the browser process.

After – Move the list of SystemMediaControlsObservers into SystemMediaControlsMac (which always runs in the browser process). RCCD will live in SystemMediaControlsBridge (which can be in or out of process). RCCD gets a mojo remote so that when it receives messages from macOS, it can tell SystemMediaControlsMac, who will notify the SMCObservers.



2.4. Duplicate PWAs

On macOS, if you open more than one "Youtube" PWA – they actually all share a single app shim. This makes sense because all Youtube PWAs are grouped under the same dock icon/Application. This however, does mean we cannot make each Youtube PWA communicate to macOS as a separate application. They must all share one communication channel to macOS. We drew inspiration from how Safari "PWAs" (called "Save to Dock" apps) behaves in this scenario and allow duplicate PWAs to steal focus from each other.

When SystemMediaControlsMac objects are about to send information across the mojo boundary to an app shim, it verifies it still holds a connection to SystemMediaControlsBridge (near (3) in the diagram)

If this connection has been severed (because some other duplicate app has stolen focus), it will re-establish the connection itself before sending messages.

When a connection is re-established, the full set of information (Album, Artist, Song, Current seek position) is automatically pushed back out to the OS.

For the reverse direction (macOS controlling the browser), whichever application has focus will receive control messages.

This is generally consistent with how Safari handles this edge case – our position is it's not currently worth additional attention due to the low expected impact of this scenario.

3. Security Considerations

This feature has some new mojo IPC connections that may have security implications. There are a few new interesting mojo IPC areas interesting to analyze from a security perspective.

3.1. Brokering the new SystemMediaControlsBridge connections

In application.mojom, we've extended the existing interface to allow for brokering of a new connection. This is not very interesting from a security perspective because it only establishes a connection between the browser process and a sub-system of the app shim which is already connected to the browser process.

3.2. The new connections

There are new mojo connections between the new bridge and its counterpart in the browser – SystemMediaControls and SystemMediaControlsObserver. These interfaces are outlined in system_media_controls.mojom.

Of these two, the SystemMediaControls interface is less interesting from a security perspective – it serves only as one-way communication from browser process to app shim process to instruct it to perform actions. If somehow, an app shim process is malicious – it is limited to not complying with the requests.

The reverse direction observer interface which allows app shims to notify the browser process of requests from macOS allows a compromised app shim to potentially spam the browser process with requests from macOS. However, this would require the browser to be actively holding a connection to that app shim – but also for it to be compromised which is unlikely.

Overall, these new connections do not seem like significant security risks.

3.3. Testing interfaces

To support testing, system_media_controls.mojom also includes a function on SystemMediaControlsObserver that allows the app shim process to notify the browser when a bridged SystemMediaControls object has been created. (see Section 8.1 – Test Approach)

4. Privacy Considerations

This feature does not pose any considerable privacy concerns due to its nature.

This feature:

- Does not store any user data.
- Does not expose additional UI not currently visible
- Does not store any preferences
- Does not interact with incognito (you cannot install apps in incognito)
- Does not fire any new telemetry

5. Prototype

5493397: [WIP][dPWA][Mac][SystemMediaControls] bridged app shim design v2 | https://chromium-review.googlesource.com/c/chromium/src/+/5493397

6. Telemetry, Flighting and Logging

6.1. Telemetry

We will wire the mac work into the same telemetry pipelines as the windows work, the metrics will be combined.

6.2. Flighting

We will follow the same procedure as our Windows work, flight to the PWA team first, then to Edge All while we wait for Chromium on-by-default.

6.3. Logging

None

7. Functional and Unit Testing

7.1. UPDATED TESTING APPROACH – 7/25/2024

Due to various blockers with multiple other testing approaches, we have shifted our testing strategy away from the methods outlined below/in comments. We will now utilize a normal browser test (not a content browser test) to install and launch PWAs, play media in them, and verify that our new bridged SystemMediaControlsBridge class gets created (in process for the browser, and out of process for the app shim).

This involves 2 main APIs:

1. `OnBridgeCreatedForTesting` – extension of system_media_controls.mojom.
   a. SystemMediaControlsBridge will use this to notify the browser process when it (ie. the bridge) has been created and its mojo connections have been set up.
2. `SetOn(Browser)BridgeCreatedCallbackForTesting` - new //content/test API.
   a. This is needed because the mojo connection between browser <-> app shim is actually between //*content*/browser <-> app shim, not //chrome/browser, where our test lives. The app shim will notify SystemMediaControlsMac, which is a private class and owned by our content::WebAppSystemMediaControlsManager, which in turn lives on content::MediaKeysListenerManagerImpl.

Our current test cases include:

- Launch 2 PWAs, play audio in both, verify 2 SystemMediaControlsBridges are made.
- Launch 1 PWA and 1 browser, play audio, verify 2 SystemMediaControlsBridges are made.
- Launch 1 browser, play audio, verify 1 SystemMediaControlsBridge is made.
- Launch 2 windows of the **same** PWA, play audio in both, verify **1** SystemMediaControlsBridge is made. (only 1 per app shim)
- Launch 1 PWA, play audio, simulate Cmd+Q quitting the app, verify nothing crashes.

7.2. Test Approach - deprecated

There are macOS specific testing challenges due to the remote app shim architecture. Our existing content_browsertests for Windows will not be adapted due to their incompatibility with the app shim model.

The primary challenge with the macOS architecture is that tests do not know how to interface with app shims. Our new mojo APIs for speaking to the app shim do not provide useful information for testing as generally they fall into two categories:

A. Set some state (e.g SetArtist) in the app shim

B. Get called back by an operation in macOS (e.g OnPlay).

We need to introduce 2 new mojo APIs for testing only:

1. Get the current state of the System Media Controls in the app shim

   This allows us to use browser_tests to spawn PWAs that play media, then query the app shim for whether the state of the playing audio is what we would expect.

2. Trigger a mocked action inside the app shim.

   This allows us to use browser_tests to trigger an action as if a user asked macOS to pause the media. An example of this is mock triggering a pause in macOS from a browser_test, which would indirectly cause the PWA media to stop playing.

7.3. Test Cases – deprecated

Our tests will be automated, with some manual verification of the end-to-end scenario.

7.3.1. Automated Test Cases

- A browser tab plays media, ensure its metadata reflects this correctly.
- 2 browser tabs playing media, ensure they still switch/fallback appropriately.
- A PWA plays media, ensure its remote app shim metadata reflects this.
- A browser tab & a PWA plays media, ensure respective metadata is correct.

- A browser tab & a PWA plays media, use mock triggers to pause one, ensure each can be controlled separately.
- 2 PWAs play media, ensure both remote app shim metadata sets reflect this.
- With 2 PWAs playing media, use the mock trigger to pause one. Ensure one is playing and one is stopped.
- 2 PWAs play media, close one – ensure it's bookkeeping has been cleaned up.
- Open a PWA, play media – ensure metadata is correctly matching this PWA. Open a duplicate of this PWA – ensure metadata has updated to reflect this PWA. Trigger an update from the original PWA by toggling paused – ensure metadata has reverted back to tracking the first PWA.

8. Known issues

8.1. macOS Now Playing state bug.

macOS has a bug in the now playing user interface when multiple apps are playing audio. This bug is not specific to Chromium, or any specific apps.

When two apps are playing media in macOS, when pausing the less recent (older) app that started playing from the macOS UI succeeds, however the macOS UI does not update to show the application paused.

This means it's not possible to resume that app, now that it is paused – as it still shows a pause button.

This bug is resolved when the other newer media playing app is interacted with, or the app triggers a state update.

This bug has been reported to Apple. We plan to proceed with this work disregarding this bug.

9. Open Questions

1. Any special handling/consideration needed for media sessions across profiles/users? For example, if you install YouTube PWA in 2 different profiles, do they each get their own app shim?
    a. Duplicate apps originating from different browser profiles still share the same app shim. This means they will behave identically to duplicate PWAs from the same profile, see Section 2.4 – Duplicate PWAs.

2. With the macOS feature flag **off**, how much of the original non-bridged design can we preserve?
    a. We will aim to keep the behavior as close to the pre-existing behavior as possible, but how that will look exactly is TBD.

10. Engineering Costs

- Add macOS specific feature flag (and figure out behavior with the flag off)
- Refactor existing `RemoteCommandCenterDelegate` component to prepare for reuse.
- Update `SystemMediaControlsMac` to support a mojo connection to a `SystemMediaControlsBridge`.
- Update `WebAppSystemMediaControlsManager` to build on macOS
- Add a `SystemMediaControls::Create` for Mac that takes an `ApplicationHost`
- Add `SystemMediaControlsBridge`
- Update `ApplicationBridge` to create `SystemMediaControlsBridge.` Add tests now that the app shim system media controls can speak to the bridge.
- Remove logging left in from Windows implementation.
- Document in-process and out-of-process bridges for SystemMediaControlsMac

11. Rejected Design Options

11.1. Use `AppShimController`/`AppShimHostMac`

[WIP][dPWA][Mac][SystemMediaControls]AppShimController/AppShimHostMac approach (5260503) · Gerrit Code Review (googlesource.com)

This approach gets the corresponding app shim by asking the AppShimManager (in chrome/browser) for the AppShimHost(Mac) given a Profile and AppId (both of which have to be obtained from WebContents). This lets us make a connection to the AppShimController (in chrome/app_shim) that corresponds to the PWA playing audio. This is more complex than just accessing the app shim via the ApplicationBridge.

This is also not ideal because it requires a content API to get from our media management code in //content/browser to the AppShimHostMac in //chrome/browser.

11.2. Remote-only bridge variant

[WIP][dPWA][Mac][SystemMediaControls] bridged app shim design v1 (5441110) · Gerrit Code Review (googlesource.com)

This variation is essentially an out of process bridge only (as opposed to in process/out of process). SystemMediaControlsBridge only runs in the app shim, and the browser's media controls connection remains largely unchanged.

This option is still viable (and actually makes the browser/in-process code simpler) but it requires a bit of code/interface duplication, as we essentially have 2 slightly different versions of some classes, 1 for the browser and 1 for the app shim.