Multi-Screen Window Placement

W3C TPAC 2022 Vancouver Update (Sept 2022) - Second Screen WG
Mike Wasserman - msw@google.com

Spec: github.com/w3c/window-placement
Demo: github.com/michaelwasserman/window-placement-demo
ChromeStatus | Oct 2021 Presentation | May 2022 vF2F Agenda & Minutes

What’s the problem?

Users of web applications are at a disadvantage on devices with multiple displays

<

Why take action?

MDN Web Developer Needs Assessment 2019

What's Missing From the Web

"Access to Hardware (12.4%)" - MDN's #1 sampled free response

"developers wanted integration with a device OS.
Also, having web apps behave more like native apps."

Overall Needs Ranking

"14. Lack of device APIs allowing for access to hardware."

The Future of the Web

"allow web applications to be more like native apps" - Respondent with 10+ yrs experience

MDN web docs
moz://a

Why take action?

MDN Web Developer Needs Assessment 2020

What's Missing From the Web:

*"Hardware/Native API. We do a lot of automation to support doctors, think **automatically position windows across multiple screens** among other things. We currently have to **install a desktop app** that the website can talk to make this work well. That combined with dictation software creates a **barrier between us and the doctors**."*

"... Configuring vsync/gsync/high refresh rate can also be frustrating."

MDN web docs
moz://a

How might we help?

CSSOM View Module
W3C Working Draft, 17 March 2016

Extend windowing APIs
for multi-screen devices

Fullscreen API
Living Standard — Last Updated 17 January 2022

HTML
Living Standard — Last Updated 12 September 2022

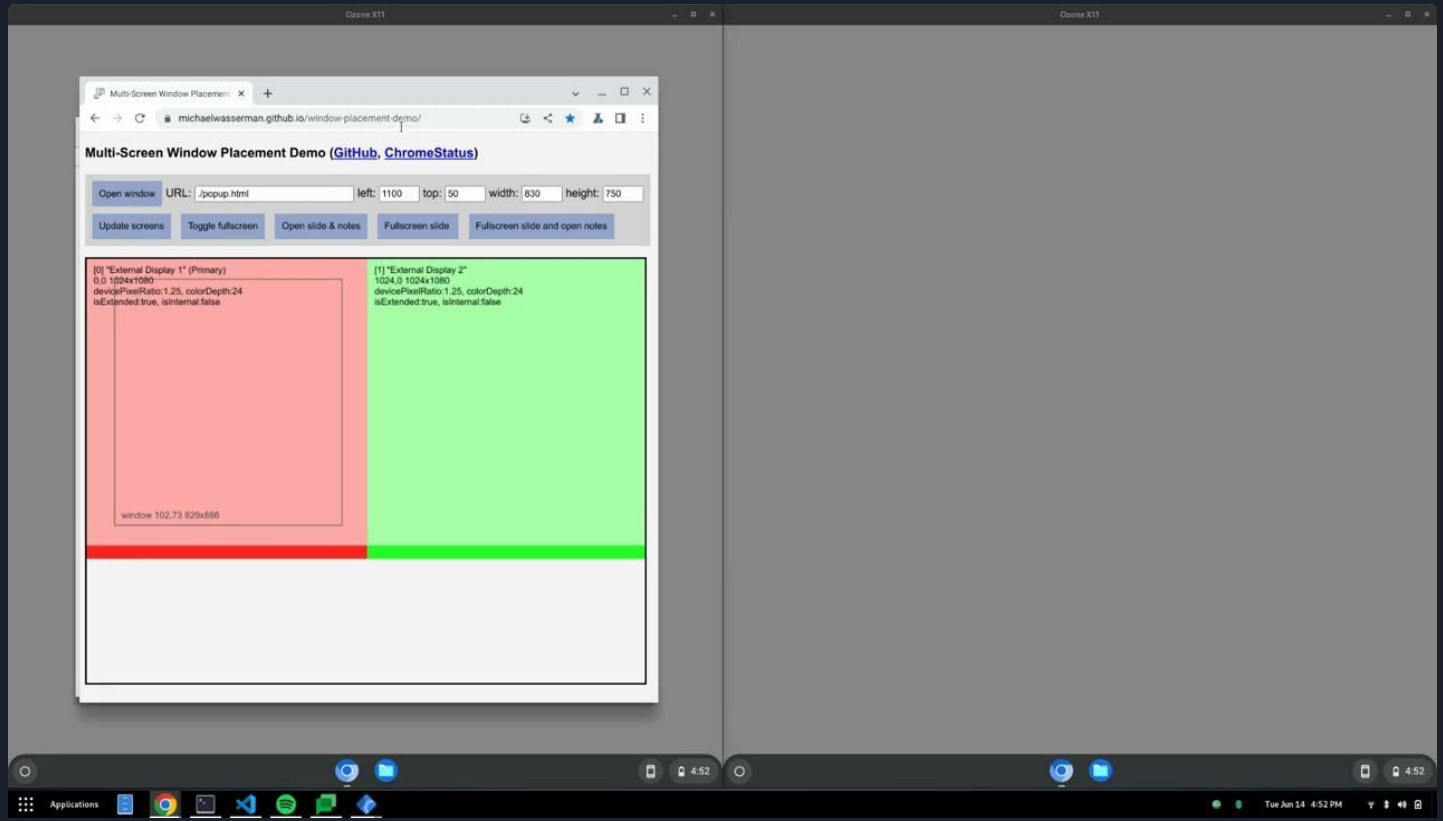
Initial info	Window.screen.isExtended indicates placement capabilities beyond Window.screen
Change events	Window.screen.onchange alleviates screen metric polling
Additional info	ScreenDetails interface provides screen info beyond Window.screen
Cross-screen placement	Screen-specific Element.requestFullscreen() & Window.open()/moveTo()/... alleviate manual window dragging
Initiating multi-screen	Multi-window extensions of user activation models alleviate repeated manual gestures

Progress Update

API FPWD published; implementation launched in Chrome M100 (ChromeStatus)

- **Fullscreen Capability Delegation** launched in Chrome M104 (ChromeStatus, Spec, Doc)
  - Allows a Window to transfer the ability to call requestFullscreen() to another
- **Fullscreen Companion Window** launched in Chrome M104 (ChromeStatus, Explainer, PR)
  - Allows sites to place fullscreen content and a popup window from one user activation
- **Accurate Screen Labels** launched in Chrome M105 (ChromeStatus)
  - Enhances screen label strings from EDIDs and higher-level OS APIs
- **HDR Support for HTMLCanvasElement** DevTrial in Chrome M105 (ChromeStatus, Explainer)
  - ColorWeb CG proposal - exposing per-screen HDR info

Let’s see a demo!



michaelwasserman.github.io/window-placement-demo/

Spec Feedback and Updates

Fullscreen Companion Window feedback and responses:

- Annevk's feedback in mozilla/standards-positions: (spoofing - fullscreen warning may go unnoticed)
- Merged PR: Clarify deceptive cross-screen placement security considerations #100
- Added Security Considerations, Example Code, security & privacy questionnaire for Explainer
  - Potential mitigation: re-show warning when fullscreen regains attention
- File a new TAG Early Design Review request and follow up on the old request

Filed issues to align and integrate spec content with established specs (premature w/o 2nd implementer):

- [cssom-view] Support for multi-screen devices #7642 (W3C/CSSWG-Drafts)
- Window object support for multi-screen devices #8217 (WHATWG/HTML)
- Proposal: Supporting fullscreen requests in multi-screen environments. #161 (WHATWG/Fullscreen)

Requested Enhancements and Speculative Explorations

Develop use cases for Initiating Multi-Screen Experiences:

- Feature request: Fullscreen support on multiple screens #92
- *“it seems like a requestFullscreen call followed by a window.open will result in the pop-up being blocked”*
- Opening child window in fullscreen / window.open should support the 'fullscreen' option #7
    - *“It would be ideal if we could open a child window in fullscreen directly as opposed to having to open it and then request fullscreen.”*
- *“multi-monitor virtual desktop software might want to open a fullscreen window on each display when the user connects to a remote or virtualized machine session”*
-

Virtual Display Testing - Intro

- **Document**: Virtual Displays For Automated Tests
- **Problem**: Lack of automated multi-screen device testing in continuous integration
- **Goal**: Add test framework support for managing high-fidelity virtual displays at the OS/WM level
  - Connect, disconnect, and reconfigure virtual displays during tests.
  - Test window creation, placement, and fullscreen on specific [virtual] screens.

Basic test pseudocode:

```
Screen.AddVirtualDisplay(<parameters>);
Window.SetBounds(<bounds on virtual display>);
EXPECT(Window.GetBounds() == <bounds on virtual display>);
Screen.RemoveVirtualDisplay(<id>);
```

Virtual Display Testing - Progress on Mac
Tremendous thanks to Fangzhen Song!

[virtual_display_mac_util.h]

virtual_display_mac_util.h

```
22 // This interface creates system-level virtual displays to support the automated
23 // integration testing of display information and window placement APIs in
24 // multi-screen device environments. It updates the displays that the normal mac
25 // screen impl sees, but not `TestScreenMac`.
26 class VirtualDisplayMacUtil : public display::DisplayObserver {
27  public:
28   VirtualDisplayMacUtil();
29   ~VirtualDisplayMacUtil() override;
30
31   VirtualDisplayMacUtil(const VirtualDisplayMacUtil&) = delete;
32   VirtualDisplayMacUtil& operator=(const VirtualDisplayMacUtil&) = delete;
33
34   // `display_id` is only used to label the virtual display. This function
35   // returns the generated display::Display id, which can be used with the
36   // Screen instance or passed to `RemoveDisplay`.
37   int64_t AddDisplay(int64_t display_id, const DisplayParams& display_params);
38   // `RemoveDisplay()` may add and remove another temporary virtual display as a
39   // workaround for known flaky timeouts awaiting the first removal of a single
40   // display.
41   // TODO(crbug.com/1126278): Resolve this defect in a more hermetic manner.
42   void RemoveDisplay(int64_t display_id);
43
44   // Check whether the related CoreGraphics APIs are available in the current
45   // system version.
46   static bool IsAPIAvailable();
```

[fullscreen_controller_interactive_browsertest.cc]

and [popup_browsertest.cc]

popup_browsertest.cc

```
255 // Tests that an about:blank popup can be moved across screens with permission.
256 IN_PROC_BROWSER_TEST_P(PopupBrowserTest, MAYBE_AboutBlankCrossScreenPlacement) {
257 #if BUILDFLAG(IS_CHROMEOS_ASH)
258   display::test::DisplayManagerTestApi(ash::Shell::Get()->display_manager())
259       .UpdateDisplay("100+100-801x802,901+100-802x802");
260 #elif BUILDFLAG(IS_MAC)
261   if (!display::test::VirtualDisplayMacUtil::IsAPIAvailable()) {
262     GTEST_SKIP() << "Skipping test for unsupported MacOS version.";
263   }
264   display::test::VirtualDisplayMacUtil virtual_display_mac_util;
265   virtual_display_mac_util.AddDisplay(
266       1, display::test::VirtualDisplayMacUtil::k1920x1080);
267 #else
268   display::ScreenBase test_screen;
269   test_screen.display_list().AddDisplay({1, gfx::Rect(100, 100, 801, 802)},
270                                         display::DisplayList::Type::PRIMARY);
271   test_screen.display_list().AddDisplay(
272       {2, gfx::Rect(901, 100, 802, 802)},
273       display::DisplayList::Type::NOT_PRIMARY);
274   display::test::ScopedScreenOverride screen_override(&test_screen);
275 #endif  // BUILDFLAG(IS_CHROMEOS_ASH)
276   display::Screen* screen = display::Screen::GetScreen();
277   ASSERT_EQ(2, screen->GetNumDisplays());
```

Virtual Display Testing - Loose plans

Enable more tests!

Support additional platforms:

- Windows: Indirect display driver shows promise
- Linux (X11): xvfb may be sufficient
- Linux (Wayland): Unsure :-/
- Chrome OS (Ash): Pre-existing mechanisms WAI
- Chrome OS (Lacros): Unsure, but I have confidence!

Unified C++ interface: TestScreenEnvironment is a step in this direction.

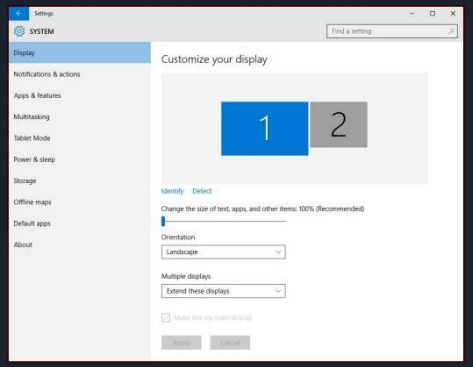
Support for WPTs: add new TestDriver wiring?

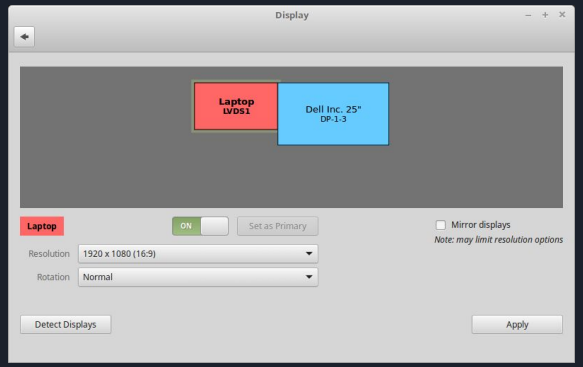
Thanks!

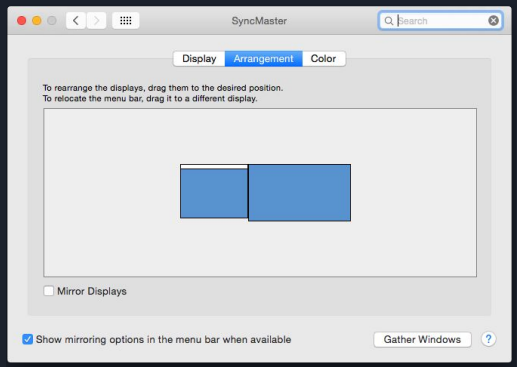
Appendix

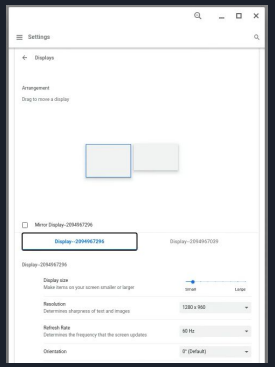
Updating the web platform for multi-screen

Existing web platform APIs	Modern Reality
Singular screen info / access	Multiple screens connected
Sync APIs with lackluster permission controls	Want capable applications with good privacy & security protections
Poor API shapes/ergonomics	Need new APIs & play nice with old







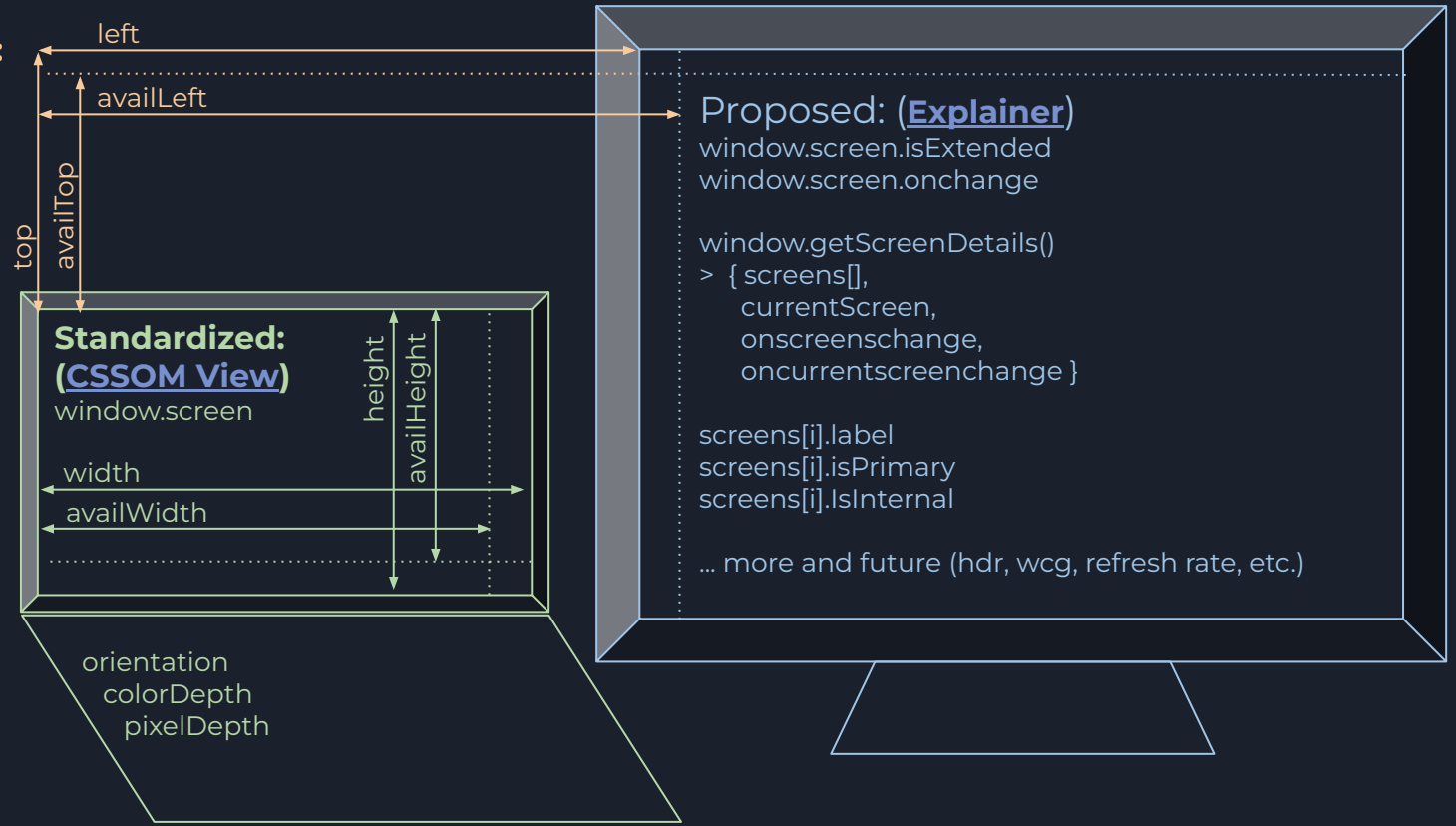


A basic example of the API usage

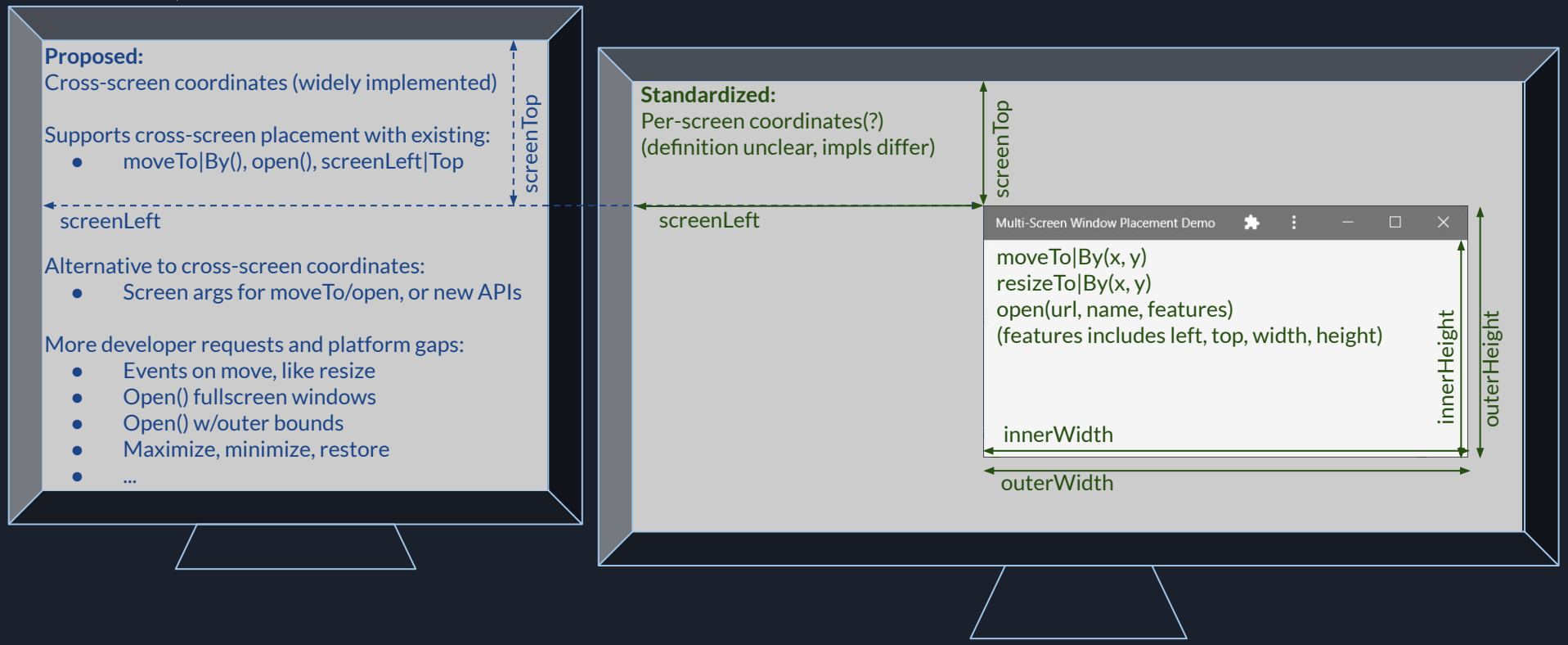
```
// Detect if the device has more than one screen.
if (window.screen.isExtended) {
  // Request information required to place content on specific screens.
  const screenDetails = await window.getScreenDetails();

  // Detect when a screen is added or removed.
  screenDetails.addEventListener('screenschange', onScreensChange);

  // Detect when the current ScreenDetailed or an attribute thereof changes.
  screenDetails.addEventListener('currentscreenchange', onCurrentScreenChange);

  // Find the primary screen, show some content fullscreen there.
  const primaryScreen = screenDetails.screens.find(s => s.isPrimary);
  await document.documentElement.requestFullscreen({screen : primaryScreen});

  // Find a different screen, fill its available area with a new window.
  const otherScreen = screenDetails.screens.find(s => s !== primaryScreen);
  window.open(url, '_blank', getWindowFeatures(otherScreen));
} else {
  // Detect when an attribute of the legacy Screen interface changes.
  window.screen.addEventListener('change', onScreenChange);
  // Arrange content within the traditional single-screen environment...
}
```

Web Platform Anatomy: Screen Info

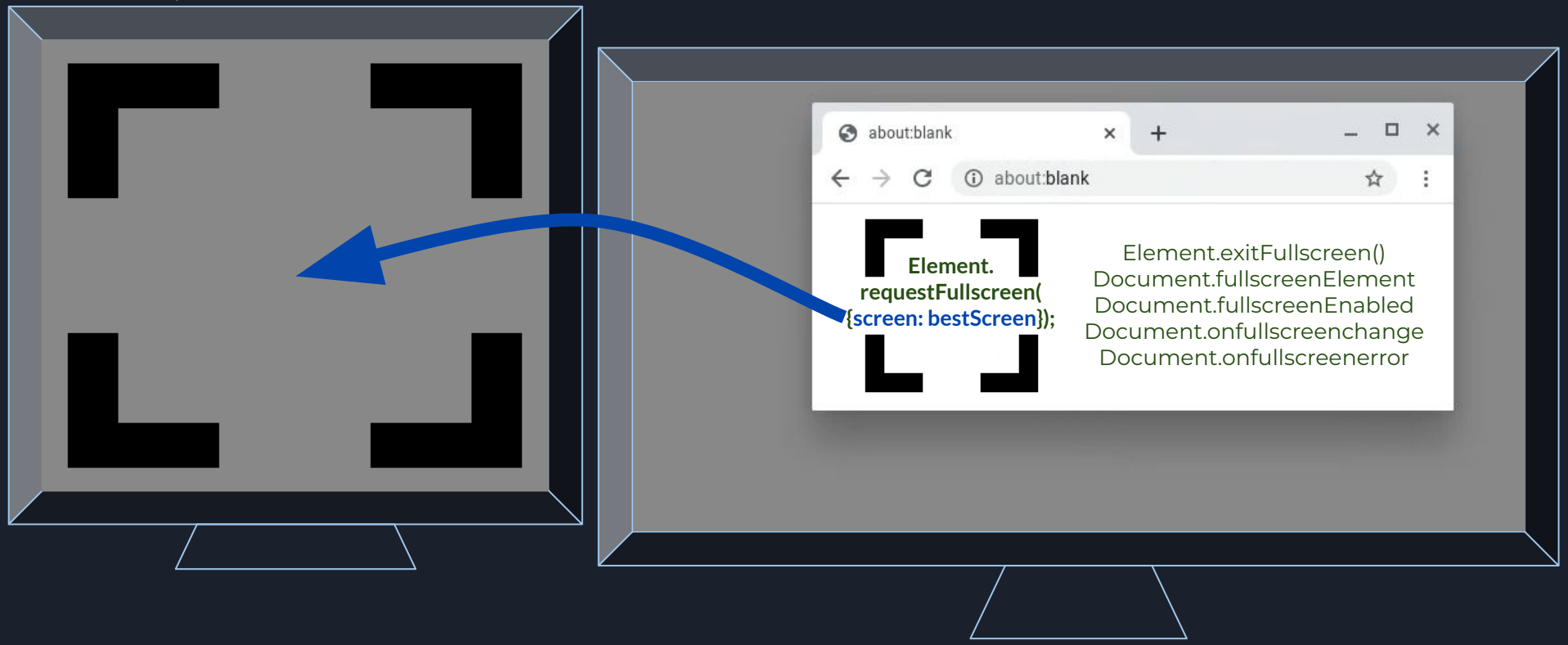
Common:
(MDN)



Web Platform Anatomy: Window Placement



Web Platform Anatomy: Fullscreen API



OT2 API Shape Changes

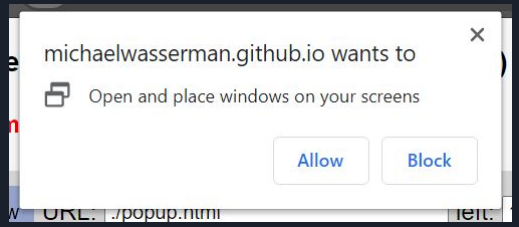
	First Origin Trial	Second Origin Trial
Are multiple screens connected?	Window.isMultiScreen() Unclear permission requirement.	window.screen.isExtended No permission required*.
Multi-screen info via Window.getScreens()	Async access to a static snapshots. Dictionary spec drifts from Screen. Need to await new info in event handlers.	Async access to live Screens interface. Exposes Screen-inheriting objects. Sync access to new info in event handlers.
Info change events	Window.onscreenschange Conflates all events. EventTarget is not gated by a permission.	Screen.onchange & Screens.on*change Per-screen & multi-screen events. EventTargets gated by permission.
Naming, etc.	screens[i].primary, screens[i].touchSupport, ...	screens[i].isPrimary, screens[i].pointerTypes, ...

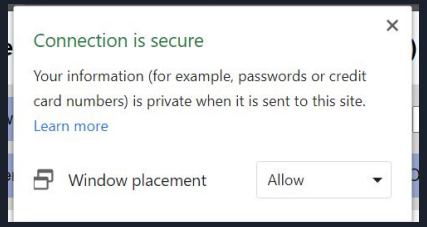
Your feedback is vital! File issues against our proposal and prototype implementation.

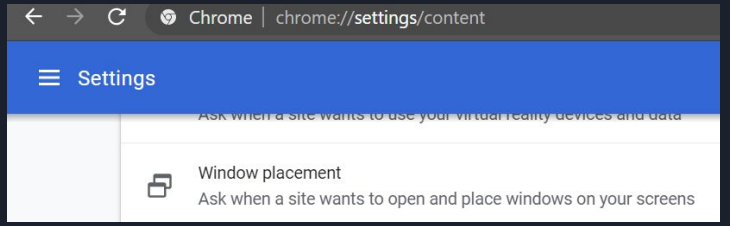
Demos

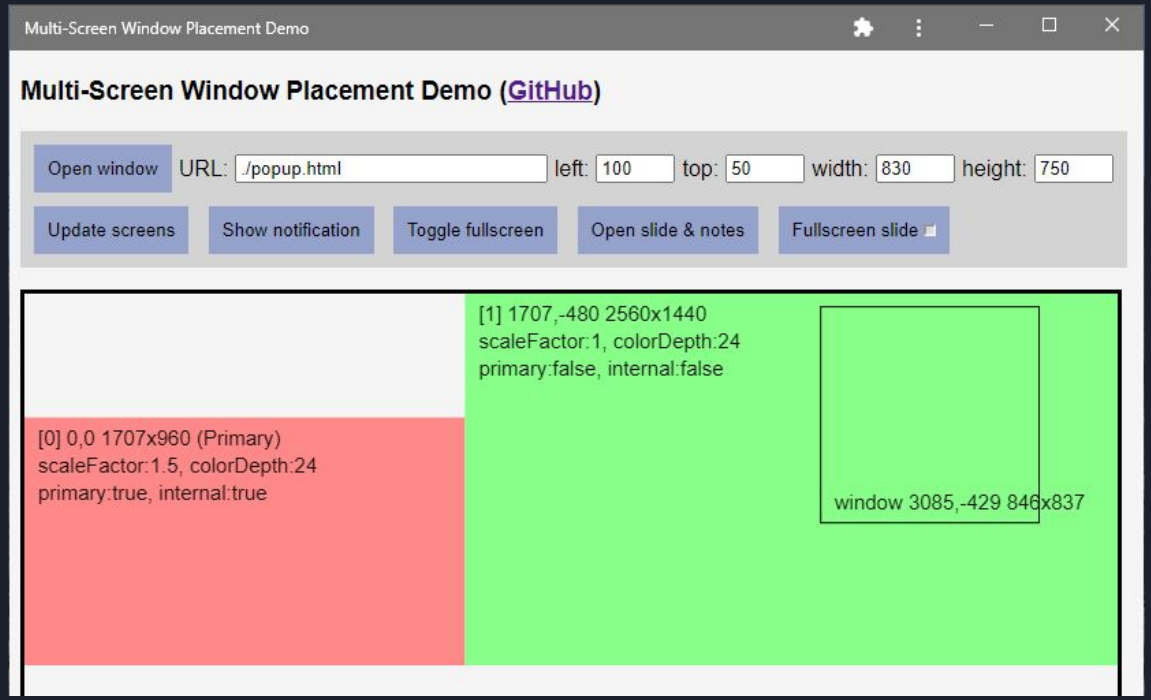
window-placement.glitch.me
web.dev/multi-screen-window-placement

michaelwasserman.github.io/window-placement-demo

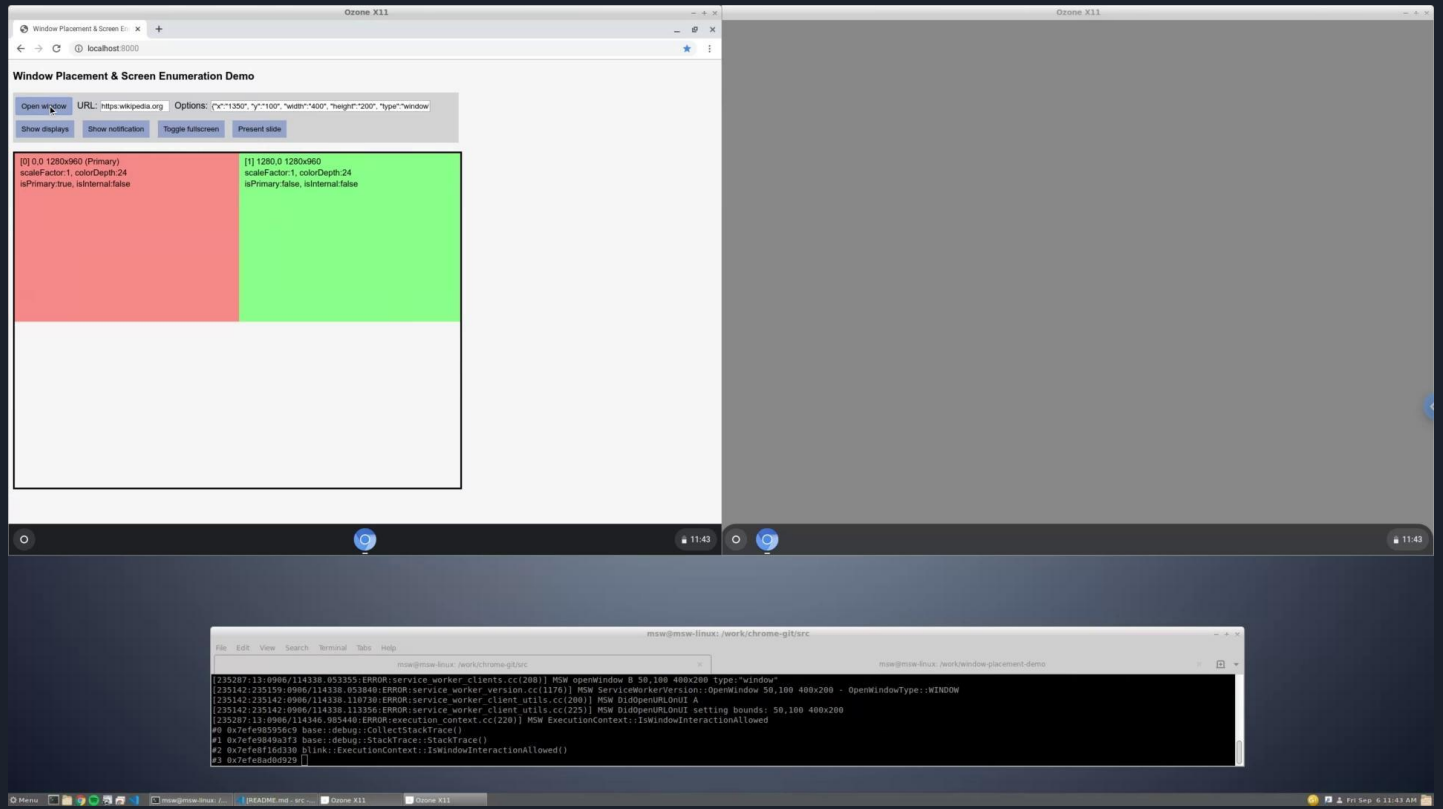








Demo


Ozone X11
Window Placement & Screen Enumeration Demo
Open window
URL: https://wikipedia.org
Options: {"x":"1350", "y":"100", "width":"400", "height":"200", "type":"window
Show displays
Show notification
Toggle fullscreen
Present slide
[0] 0,0 1280x960 (Primary)
scaleFactor:1, colorDepth:24
isPrimary:true, isInternal:false
[1] 1280,0 1280x960
scaleFactor:1, colorDepth:24
isPrimary:false, isInternal:false
Ozone X11
msw@msw-linux: /work/chrome-git/src
[235287:13:0906/114338.053355:ERROR:service_worker_clients.cc(208)] MSW openWindow B 50,100 400x200 type:"window"
[235142:235159:0906/114338.053840:ERROR:service_worker_version.cc(1176)] MSW ServiceWorkerVersion::OpenWindow 50,100 400x200 - OpenWindowType::WINDOW
[235142:235142:0906/114338.110730:ERROR:service_worker_client_utils.cc(200)] MSW DidOpenURLOnUI A
[235142:235142:0906/114338.113356:ERROR:service_worker_client_utils.cc(225)] MSW DidOpenURLOnUI setting bounds: 50,100 400x200
[235287:13:0906/114346.985440:ERROR:execution_context.cc(220)] MSW ExecutionContext::IsWindowInteractionAllowed
#0 0x7efe985956c9 base::debug::CollectStackTrace()
#1 0x7efe9849a3f3 base::debug::StackTrace::StackTrace()
#2 0x7efe8f16d330 blink::ExecutionContext::IsWindowInteractionAllowed()
#3 0x7efe8ad0d929

Feedback? Questions? Let’s chat!

- API refinements
- Related proposals
- Implementation quirks
- Additional platform gaps
- And more ...

Spec issues: github.com/w3c/window-placement

Impl issues: crbug.com (component:Blink>Screen>MultiScreen)

Contacts: msw@ or desktop-pwas-team@

Integration with related APIs/proposals

Window Segments Enumeration API

- Exposes bounds for each content region of a single window that spans multiple (?) Screens
  - *partial interface Window { sequence<DOMRect> getWindowSegments(); };*
- If one Screen can yield multiple segments, should per-Screen segments be exposed? (issue #7)
  - Expose which screens have segments/folds before a window is placed there?
  - Add *partial interface Screen { readonly attribute FrozenArray<DOMRect> segments; };*?

Screen Fold API

- Exposes the angle and orientation of a fold in a single (?) Screen
  - *partial interface Screen { [SameObject] readonly attribute ScreenFold fold; };*
- Support for: One fold between two Screens? Off-center folds? Multiple folds per Screen? (issue #38)
  - Add *partial interface ScreenFold { readonly attribute FrozenArray <Screen> screens; };*? More?
  - Add *partial interface ScreenFold { readonly attribute long position; };*?
  - Use *partial interface Screen { [SameObject] readonly attribute FrozenArray<ScreenFold> folds; };*?

Visual Viewport API

- Exposes information about the scaling and scrolling of content within a Window

See related Multi-Screen Window Placement issues #21, #35, #36
Naive principle: A multi-screen API should expose all Screen interface info for each available Screen.
For example: if *Screen.hdr* was added, one should expect this to work: *(await getScreens()).screens[i].hdr;*

Coordinate system standardization

Spec is unclear about multi-screen environments; let's consider options...

	Cross-screen window coordinates New placement APIs not needed	**Per-screen window coordinates** New placement APIs needed
Maximize Privacy	UA lies, e.g. { Window.screenLeft|Top,Screen.Width|Height == 0, Window.outerWidth|Height,Screen.width|height == Window.innerWidth|Height} Window placement isn't really feasible; fingerprinting is minimized...	
Toggle with permission?	UA lies w/o permission; gives actual coordinates w/permission ScreenLeft|Top & outerWidth|Height change w/permission... unprecedented?	
Maximize Transparency	UA gives actual coordinates, other multi-screen info gated by permission windows on separate screens can collude, e.g. win1.screen.width != win2.screen.width	
	Sites w/o permission can sometimes infer multi-screen geometry with one window. (e.g. screenLeft > screen.width)	Sites w/o permission can't infer multi-screen geometry with one window. Sites w/o permission can't always discern if two windows are on the same display.

Cross-screen fullscreen window behavior

Chromium uses the underlying window for fullscreen (browser/popup/web application).
So, the cross-screen fullscreen prototype moves the underlying window to the target screen.

Users may perceive that the window has “disappeared” while an element is fullscreen.

Is this purely an implementation detail? Should the Fullscreen API prescribe behavior?

Also, should we support multiple fullscreen elements from a single document?

Looking ahead: Possible future proposals

Here are some developer requests and platform gaps not (yet) addressed by this proposal.

Window placement:

- Events on move, like resize (exploration)
- Open() fullscreen windows (#7)
- Open() w/outer bounds
- Maximize, minimize, restore, focus (#3)
- Moving/swapping fullscreen screens (#5)
- Z-ordering… :-/ (#10)
- More ergonomic and powerful APIs… (#8, explorations: A, B)
- Parent/child and modal window relationships? (exploration)
- Other properties (exploration)

Screen information:

- HDR & WCG
- Refresh Rate
- Other info (exploration)

Chromium Implementation Anatomy

Browser

RenderFrameHost (&View&Widget)
ScreenEnumerationImpl Mojo service impl
UpdateVisualProperties & ScreenInfo legacy IPC

PermissionControllerImpl (per-frame), Activation
exclusive access FullscreenController

WebContentsImpl::RequestSetBounds
WebContentsImpl::ShowCreatedWindow
Browser::AddNewContents, navigation, initial_bounds

unit/browser/interactive_ui tests, UMA

views::Widget bounds init clamping & WindowSizer
display::Screen[Ash|Base|Mac|Ozone|Win|X11]
ui/display::Display & display.mojom

Renderer

RenderFrame (&View&Widget)
GlobalScreenEnumeration Mojo client
UpdateVisualProperties & ScreenInfo legacy IPC

permission.mojom, *_descriptor.idl, *_util.cc
Blink-side FullscreenController

[Local]DomWindow & Screen JS interface impls
ChromeClient::SetWindowRectWithAdjustment
IDLs: screen, window, fullscreen & *_options

Web platform tests, UseCounter

Subframe FeaturePolicy, permission delegation
Execution context lifetimes, promises, async fun