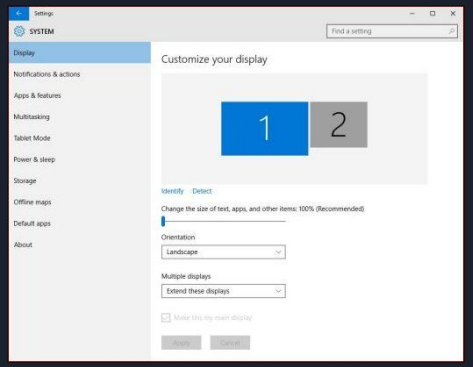
Multi-Screen Window Placement

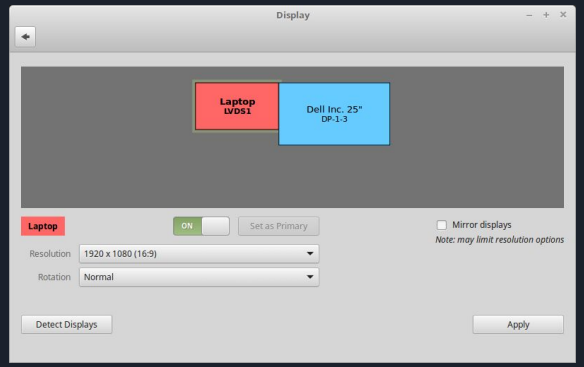
TPAC 2020 - Second Screen CG
Incubation overview & update
msw@google.com

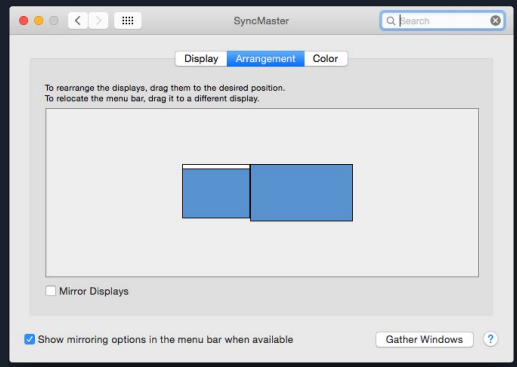
Draft Spec/Explainer/Issues: github.com/webscreens/window-placement
Chromium OT M86-M88: ChromeStatus entry, origin trials tokens, meta bug

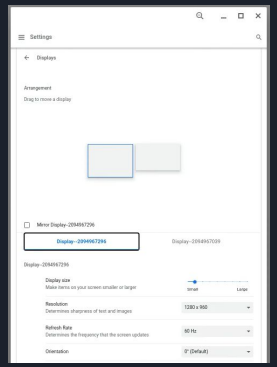
Updating the web platform for multi-screen

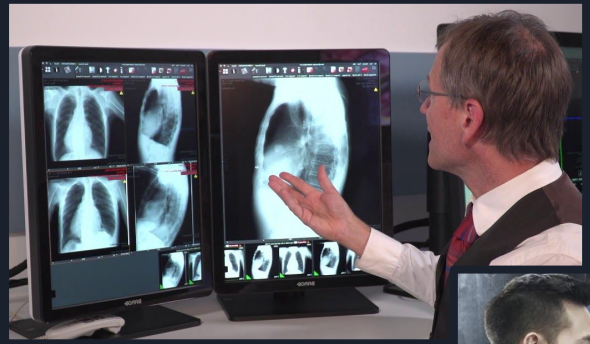
Existing web platform APIs	Modern Reality
Singular screen info / access	Multiple screens connected
Sync APIs with lackluster permission controls	Want capable applications with good privacy & security protections
Poor API shapes/ergonomics	Need new APIs & play nice with old

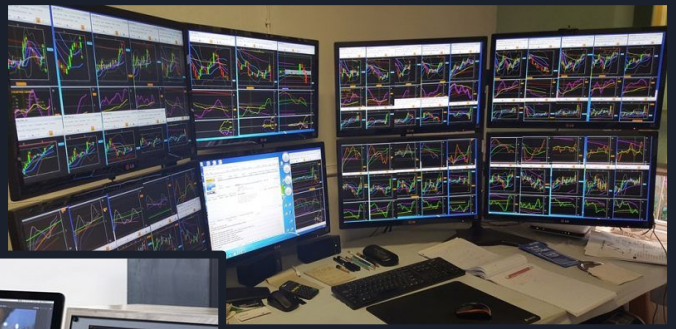













The more you shop, the more you save.


Computer Hope

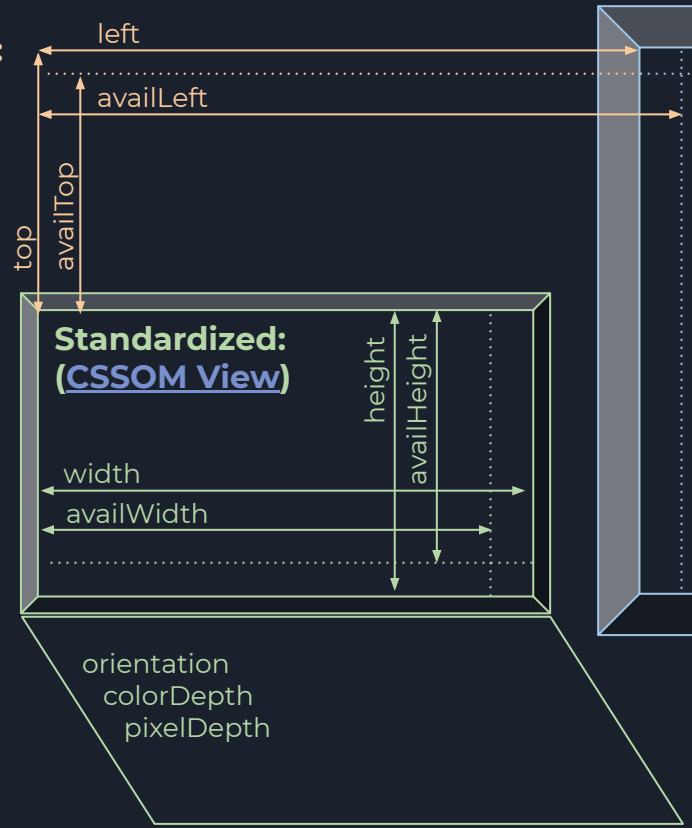
Updating web APIs for multi-screen

	Current	Proposal
Initial information	Just the current window.screen	New multi-screen boolean
Change handling	Sites must poll for changes	Events fire on screen changes
Additional information	No :-/	Request other screens and additional per-screen info
Cross-screen fullscreen	Users must manually drag windows, then trigger fullscreen...	Sites can specify a screen or swap to another screen
Cross-screen windows	Users must manually drag windows (bounds clamped to current screen*)	Sites can specify cross-screen coordinates for window.open()/moveTo()/...

Get a Chromium Origin Trial (M86-M88) token and test it today!

Web Platform Anatomy: Screens

Common:
(MDN)



Proposal:
window.screen.isExtended
window.screen.onchange

window.getScreens()
> { screens[], currentScreen, onchange }
screens[i].id
screens[i].isPrimary
screens[i].IsInternal
screens[i].devicePixelRatio
screens[i].pointerTypes

More developer requests and platform gaps:

- Screen name, hdr, wcg, etc.
- ...

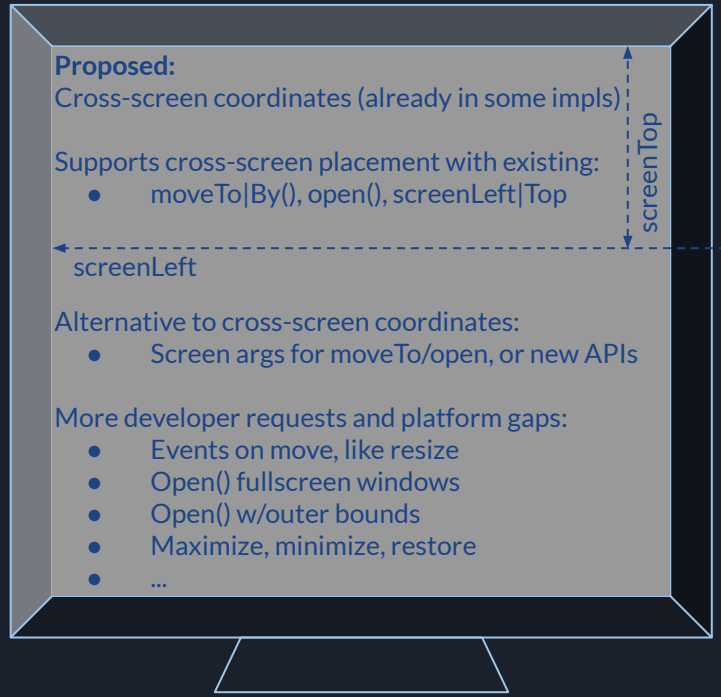
Web Platform Anatomy: Window Placement

Proposed:
Cross-screen coordinates (already in some impls)

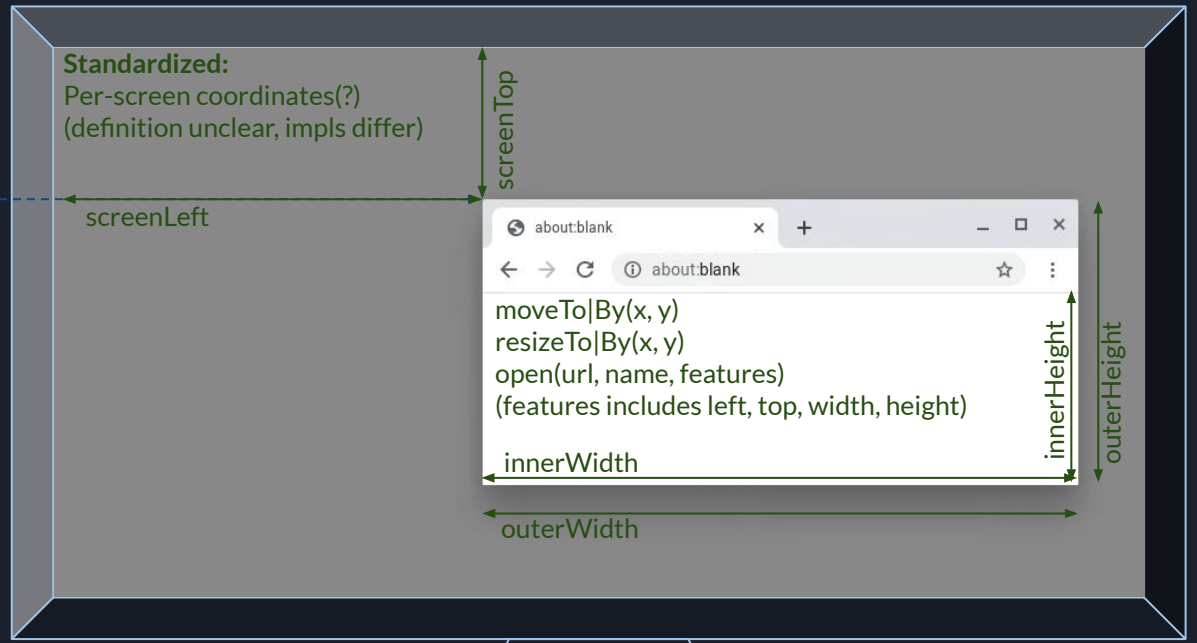
Supports cross-screen placement with existing:
- moveTo|By(), open(), screenLeft|Top

Alternative to cross-screen coordinates:
- Screen args for moveTo/open, or new APIs

More developer requests and platform gaps:
- Events on move, like resize
- Open() fullscreen windows
- Open() w/outer bounds
- Maximize, minimize, restore
- ...



Standardized:
Per-screen coordinates(?)
(definition unclear, impls differ)



Related: Element.requestFullscreen({screen: bestScreen});

Ongoing API Shape Changes

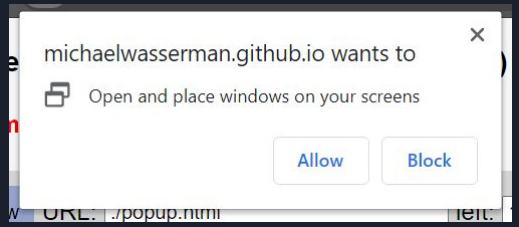
	First Origin Trial	Proposed Update
Are multiple screens connected?	Window.isMultiScreen() Unclear permission requirement.	window.screen.isExtended No permission required*.
Multi-screen info via Window.getScreens()	Async access to a static snapshots. Dictionary spec drifts from Screen. Need to await new 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.onchange Per-screen & multi-screen events. EventTargets gated by permission.
Naming, etc.	screens[i].primary, screens[i].touchSupport, ...	screens[i].isPrimary, screens[i].pointerTypes, ...

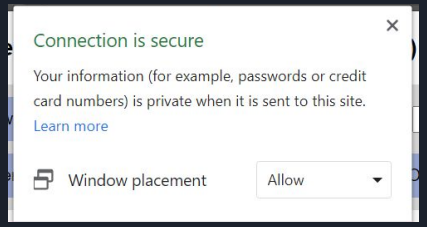
Your feedback is vital! File issues against our proposal and prototype implementation.

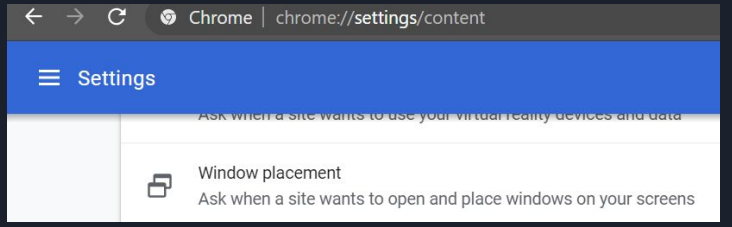
Demo

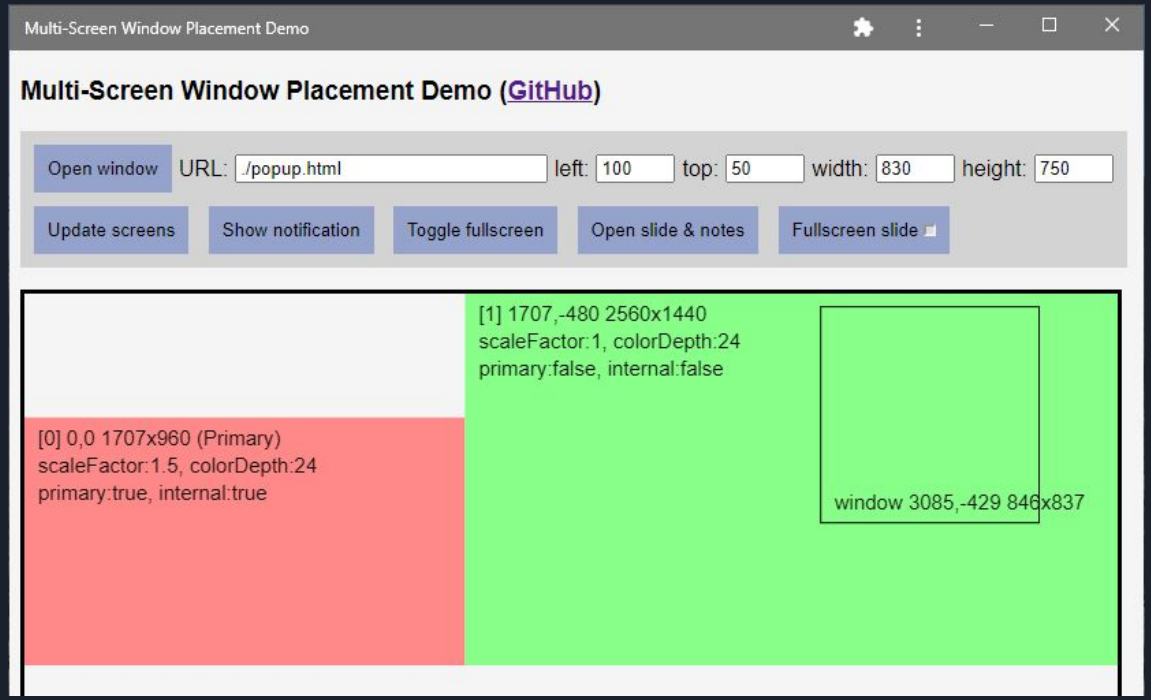
window-placement.glitch.me
web.dev/multi-screen-window-placement

michaelwasserman.github.io/window-placement-demo

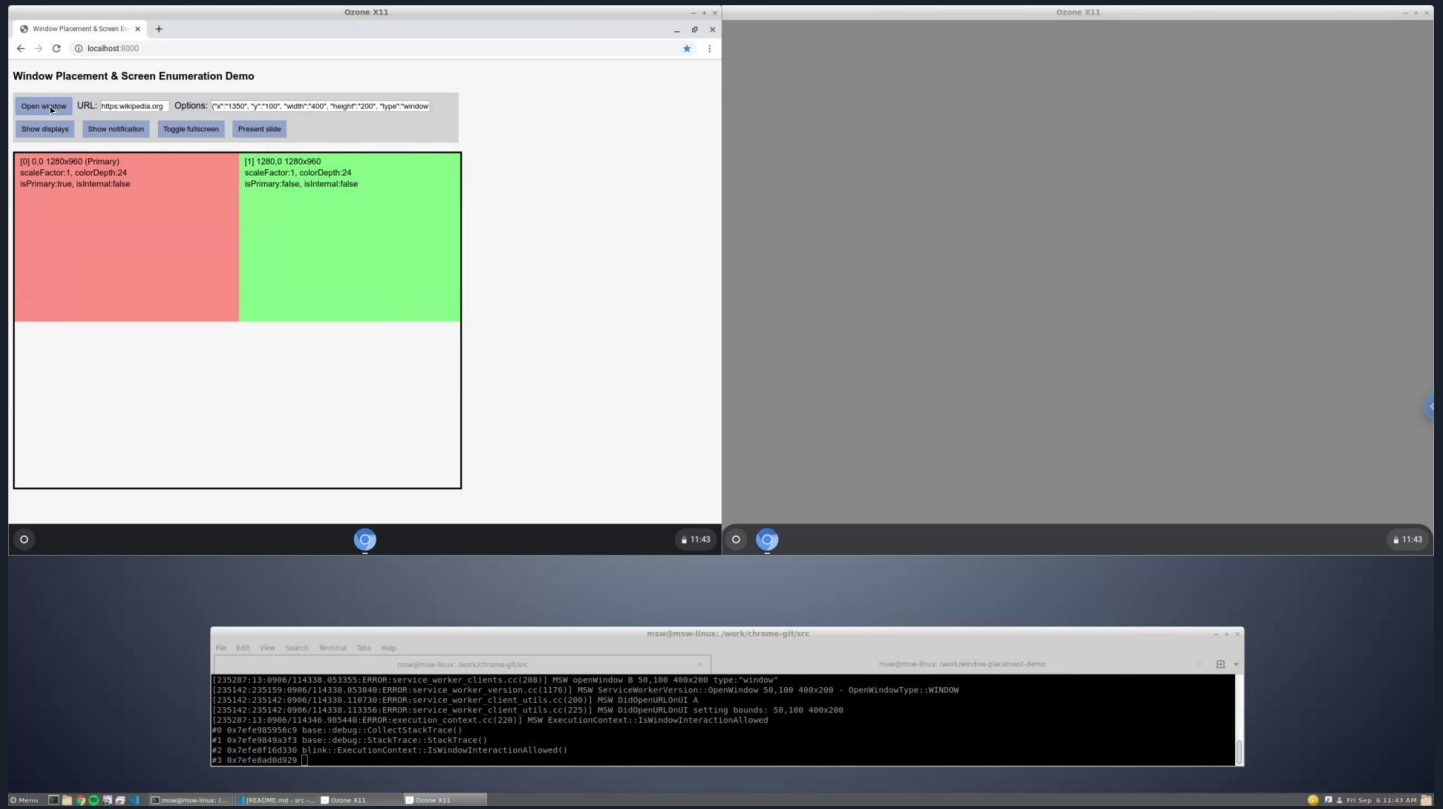








Demo


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

Thank you!

Thanks! Questions? Comments?

Possible discussion topics:

- Integration with related APIs/proposals
- Coordinate system standardization
- Cross-screen fullscreen window behavior
- Naming/shaping the current API :-)
- Looking ahead: Possible future proposals

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?

Naming/shaping the current API :-)

Screen.isExtended
Screen.onchange
Window.getScreens()

```
interface Screens : EventTarget {
  readonly attribute FrozenArray<ScreenAugmented> screens;
  readonly attribute ScreenAugmented currentScreen;
  attribute EventHandler onchange;
};

interface ScreenAugmented : Screen {
  readonly attribute long left;
  readonly attribute long top;
  readonly attribute long availLeft; // On Screen...
  readonly attribute long availTop; // On Screen...
  readonly attribute boolean isPrimary;
  readonly attribute boolean isInternal;
  readonly attribute float devicePixelRatio;
  readonly attribute DOMString id;
  readonly attribute FrozenArray<PointerType> pointerTypes;
};
```

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:

- Name
- HDR support
- WCG support
- 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

Sounds easy! Let's just...

- Share use cases and explain value, refine API shapes, gather feedback and interest
- Intent to Prototype + basic dev-trial implementation
  - Plumb Browser's screen info to Blink via Mojo; expose via IDL
  - Extend fullscreen IDL for a requested screen, plumb to Browser
  - **Retrofit scattered and buggy window bounds clamping in Blink/Views**
- Write a demo PWA, keep it up to date with API changes and highlighted use cases
- Add permission prompt, checks, page info bubble & WebUI settings entries
  - Determine permission text, icons, one permission or two...
  - Weigh partner requests against privacy objectives
  - Plumb source RenderFrameHost/permissions through new window creation, etc.
- Continually refine API scope, shape, and implementation details
  - Investigate/fix per-platform bugs (e.g. workspace transitions on Mac, per-screen scaling on Win)
  - Add per-platform support (e.g. internal display detection)
  - Explore subframe feature policy capabilities and implications
  - Extend browser/wpt tests for multi-screen (w/o multi-screen hardware), etc.
  - Scratch head over window.open()'s lack of outerWidth & outerHeight support…
  - Work around transient user gesture expirations during permission prompts
  - Implement metrics somewhere in the stack of requests and outcomes
- Coordinate with W3C TAG review, partners, related proposals
  - **Address feedback, improve explainer materials, organize alternatives and V2 explorations**
- Soon: Intent to Experiment + Origin Trial! -> Refine, Intent to Ship, Launch&Land, Standardize

Thanks!