WebXR DOM Overlays Module

W3C Working Draft, 24 September 2024

▼ More details about this document

This version:
https://www.w3.org/TR/2024/WD-webxr-dom-overlays-1-20240924/

Latest published version:
https://www.w3.org/TR/webxr-dom-overlays-1/

Editor's Draft:
https://immersive-web.github.io/dom-overlays/

Previous Versions:
https://www.w3.org/TR/2022/WD-webxr-dom-overlays-1-20221117/

History:
https://www.w3.org/standards/history/webxr-dom-overlays-1/

Feedback:
GitHub

Editor:
Piotr Bialecki (Google)

Former Editor:
Klaus Weidner (Google)

Participate:
File an issue (open issues)
Mailing list archive
W3C's #immersive-web IRC

▶ **Unstable API**

Abstract

The WebXR DOM Overlays module expands the WebXR Device API with a mechanism for showing interactive 2D web content during an immersive WebXR

session. When the feature is enabled, the user agent will display the content of a single DOM element as a transparent-background 2D rectangle.

Status of this document

This section describes the status of this document at the time of its publication. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

The Immersive Web Working Group maintains a list of all bug reports that the group has not yet addressed. This draft highlights some of the pending issues that are still to be discussed in the working group. No decision has been taken on the outcome of these issues including whether they are valid. Pull requests with proposed specification text for outstanding issues are strongly encouraged.

This document was published by the Immersive Web Working Group as a Working Draft using the Recommendation track. This document is intended to become a W3C Recommendation.

Publication as a Working Draft does not imply endorsement by W3C and its Members. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 03 November 2023 W3C Process Document.

Table of Contents

§ 1. Introduction

This section is non-normative.

This module describes a mechanism for showing interactive 2D web content during an immersive WebXR session. When the feature is enabled, the user agent will display the content of a single DOM element as a transparent-background 2D rectangle.

1.1. Overview

This section is non-normative.

While the DOM overlay is active, the UA enables user interactions with the DOM overlay's content using platform-appropriate mechanisms. For example, when using XR controllers, the primary action dispatches DOM pointer events and click events at the location where the controller's pointing ray intersects the DOM overlay.

A new beforexrselect event provides a way to suppress XR input events for specific regions of the DOM overlay and helps applications distinguish DOM UI interactions from XR world interactions.

2. HTML API Integration

This module adds a new event type to the definition of `GlobalEventHandlers`.

2.1. onbeforexrselect

An `XRSessionEvent` of type ***beforexrselect*** is dispatched on the DOM overlay element before generating a WebXR `selectstart` input event if the -Z axis of the input source's `targetRaySpace` intersects the DOM overlay element at the time the input device's primary action is triggered.

```
partial interface mixin GlobalEventHandlers {
  attribute EventHandler onbeforexrselect;
};
```

This event is an `XRSessionEvent` with type beforexrselect that bubbles, is cancelable, and is composed. Its `target` element is the topmost event target being intersected by the `targetRaySpace` and is either a descendant of the DOM overlay element or the DOM overlay element itself.

Cancelling this event by calling `preventDefault()` suppresses default WebXR input events that would normally be generated by the input source for this primary action. The `selectstart`, `selectend`, and `select` events will not be fired for this interaction sequence.

NOTE: Future WebXR modules MAY define additional events or WebXR input dependent data that are affected by cancelling this event, for example suppressing results from a transient input source's hit test subscription.

This event and the actions taken by the event handler have no effect on DOM event processing, and are not synchronized with DOM event dispatch. The user's action will separately generate appropriate DOM events such as `"pointerdown"`, and those DOM events can happen before or after the corresponding beforexrselect event. This happens regardless of whether the beforexrselect event was cancelled or not, and is independent of any further actions taken in XR input event handlers.

NOTE: This event provides a way for applications to suppress duplicate XR input events while the user is interacting with a DOM UI. Since this is a bubbling event, the application can register handlers on appropriate container elements, effectively marking regions of the DOM overlay as blocking XR input. This is independent of the visual opacity of DOM elements. It is possible to show noninteractive opaque or translucent DOM content such as text explanations that don't block XR input events.

EXAMPLE 1 ¶

The following code installs an event handler on an interactive part of the DOM overlay to selectively suppress XR events for that region, while continuing to generate XR events for other parts of the DOM overlay that are treated as transparent for interaction purposes.

```
document.getElementById('button-container').addEventListener(
  'beforexrselect', ev => ev.preventDefault());
```

§ 2.2. CSS pseudo-class

The *':xr-overlay'* pseudo-class MUST match the overlay element for the duration of an immersive session using a DOM Overlay.

The overlay element is a backdrop root.

NOTE: Backdrop filter effects on the DOM overlay element or its descendants do not modify the AR camera image (if applicable) or the rendered content drawn to the immersive session's `XRWebGLLayer`.

The stacking contexts for ancestors of the overlay element, if any, do not paint to the immersive session's display.

> NOTE: The overlay element itself is a stacking context due to `position: fixed` styling.

> NOTE: on a multi-display system, the UA MAY paint and draw stacking contexts for ancestors or sibling trees of the overlay element on a separate display such as a desktop monitor.

§ 2.3. User-agent level style sheet defaults

The user-agent style sheet defaults for the overlay element are as follows:

```
:xr-overlay {
    /* force a transparent background */
    background: rgba(0,0,0,0) !important;

    /* act as containing block for descendants */
    contain: paint !important;

    /* the following styling is identical to :fullscreen */
    position: fixed !important;
    top: 0 !important;
    right: 0 !important;
    bottom: 0 !important;
    left: 0 !important;
    margin: 0 !important;
    box-sizing: border-box !important;
    min-width: 0 !important;
    max-width: none !important;
    min-height: 0 !important;
    max-height: none !important;
    width: 100% !important;
    height: 100% !important;
    transform: none !important;

    /* intentionally not !important */
    object-fit: contain;
}
```

NOTE: This is based on *Fullscreen API* § 5.2 User-agent level style sheet defaults, with additional styling to make the overlay element's background transparent. The styling for :xr-overlay does not explicitly depend on the Fullscreen API's pseudoclass or styling so that user agents have the flexibility to implement it independently of the Fullscreen API.

NOTE: The Fullscreen API does not currently specify the `contain: paint` rule, though this matches typical UA behavior and is planned to be added in a future revision of that specification.

NOTE: Applications are encouraged to use the :xr-overlay pseudo-class for conditionally styling UI elements during the session, including controlling visibility of interface elements.

§ 2.4. Fullscreen API integration

The UA MAY implement DOM Overlay as a special case of the [FULLSCREEN] API. In this case, the UA MUST prevent changes to the active fullscreen element, rejecting `requestFullscreen` requests for the duration of the immersive session.

NOTE: The DOM Overlay API requires specifying the overlay element at session start, and does not provide a mechanism to change the active overlay element during the session. Applications would behave inconsistently across platforms if they could use the Fullscreen API to indirectly change the active overlay element.

When DOM Overlay is implemented through the Fullscreen API, the `root` element stacking context does not paint to the immersive display. Only the stacking contexts for the elements in the top layer, including the overlay element, paint to the immersive display.

NOTE: By default, fullscreen mode uses an opaque black backdrop. The modified paint rules ensure that this backdrop does not need to be drawn, and that ancestors of the overlay element or sibling trees aren't visible through the transparent overlay element.

NOTE: Allowing implementation based on the Fullscreen API is primarily intended for single-display systems where the rest of the page is not visible during the immersive session. A multi-display system could technically use the Fullscreen API for the overlay element while showing the rest of the page on a separate display such as a desktop monitor, and in that case the UA MAY paint and draw stacking contexts for ancestors or sibling trees of the overlay element on the separate display.

Alternatively, the UA MAY implement DOM Overlay independently of the [FULLSCREEN] API. In this case, the overlay element MUST still match the :xr-overlay pseudoclass and MUST be styled in the immersive view using the § 2.3 User-agent level style sheet defaults for this pseudoclass. The UA MAY separately support using the fullscreen API for elements outside the overlay element, but this MUST NOT have any effect on how the DOM overlay content is displayed.

NOTE: Handling DOM Overlay and Fullscreen API independently is intended to support a multi-display system such as a desktop PC with an attached VR headset. In this case, the Fullscreen API could be used to control page content on the 2D monitor, for example showing a fullscreen canvas element with a third-person rendered view, while the DOM overlay element and immersive content is separately displayed in the headset.

On a multi-display system where the immersive session uses a separate output device from the originally displayed web page, the overlay element MUST NOT be visible or interactive on other displays as part of a 2D web page while it is being shown in the immersive view. The UA MAY choose to hide or disable the entire page on other displays for the duration of the session.

NOTE: It is OK to show the DOM overlay content as part of a non-interactive headset mirror view or similar non-web-page UI. The intent of this multi-display restriction is to avoid inconsistent display of the overlay element and potentially confusing interactions if it's shown in two places at once. This also avoids implementation challenges related to displaying a DOM element on two separate displays simultaneously.

§ 3. WebXR Device API Integration

This module expands the definitions of `XRSessionInit` and `XRSession`, and modifies the behavior of `XRInputSource` events.

§ 3.1. XRSessionInit

This module introduces the string ***dom-overlay*** as a new valid feature descriptor for use in the `requiredFeatures` or `optionalFeatures` sequences for immersive sessions.

A device is capable of supporting the DOM overlay feature if it provides a way for the user to see and interact with DOM content for the duration of the immersive session.

> NOTE: Implementation choices include a fullscreen overlay on a handheld AR device, or a floating rectangle in space for a VR or AR headset.

The DOM content MUST be composited as if it were the topmost content layer. It MUST NOT be occluded by content from the `XRWebGLLayer` or by images from a passthrough camera for an AR device. Applications can use normal CSS rules to control transparency and 2D placement of content within the DOM overlay itself.

The DOM overlay MUST be automatically visible to the user from the start of the session, without requiring the user to press buttons or take other manual actions to make it visible.

> NOTE: A device should not claim to support a DOM overlay if the content element is only indirectly visible, for example if the user would need to take off their headset or manually enable a passthrough camera to view content on a separate 2D monitor that's not normally visible during the session. However, an immersive CAVE system where a user is carrying a physical touchscreen device showing the DOM overlay content would be a valid implementation.

The XRSessionInit dictionary is expanded by adding a new `domOverlay` member. This is an optional member of `XRSessionInit`, but it MUST be specified when using the DOM overlay feature since there is no default overlay element.

```
partial dictionary XRSessionInit {
  XRDOMOverlayInit? domOverlay;
};
```

If the DOM overlay feature is a required feature but the application did not supply a `domOverlay` member, the UA MUST treat this as an unresolved required feature and reject the `requestSession()` promise with a `NotSupportedError`. If it was requested as an optional feature, the UA MUST ignore the feature request and not enable a DOM overlay.

NOTE: The UA MAY emit local warnings such as developer console messages explaining why the DOM overlay was not enabled.

§ 3.2. XRSession

This module extends the XRSession interface to add a new readonly attribute which reflects the current state of the DOM overlay feature.

```
partial interface XRSession {
  readonly attribute XRDOMOverlayState? domOverlayState;
};
```

The ***domOverlayState*** attribute MUST be null if the dom-overlay feature is not supported or not enabled.

If the feature is enabled, the attribute value MUST be present.

NOTE: Applications can check the presence `domOverlayState` to verify that the DOM overlay feature is enabled and working, for example if it was requested as an optional feature.

NOTE: The DOM overlay may be temporarily invisible to the user, for example if the user agent places it at a fixed orientation or location where it may end up outside the user's field of view after user movement. The `domOverlayState` attribute still remains set while this is happening.

While the session is active with a visible DOM overlay, the UA MUST treat this as rendering opportunity and execute `Window` `requestAnimationFrame()` callbacks at a rate suitable for animating DOM content. These MAY run at different times and frequencies than `requestAnimationFrame()` callbacks as used for drawing `XRWebGLLayer` content.

§ 3.3. XRInputSource

When an `XRInputSource` begins the platform-specific action corresponding as its primary action the UA MUST run the following steps before starting input processing to decide if this is treated as a primary action:

1. If the input source's `targetRaySpace` intersects the DOM overlay at the time the input device's primary action is triggered:

    1. Queue a task to fire an event named beforexrselect using `XRSessionEvent` on the topmost event target within the DOM overlay `root` being intersected by the `targetRaySpace`, setting `target` to that element. This events bubbles, is cancelable, and is composed.
    2. Check how XR input should be handled as follows:

    ↪ If the event was cancelled

        1. If the input source is a transient input source, treat this as an auxiliary action. Otherwise, ignore this action for the purpose of generating XR input events.

    ↪ Otherwise

        Treat the action as a primary action as usual for the input source.

NOTE: Effectively, cancelling the beforexrselect event suppresses XR input select events, none of `selectstart`, `selectend`, or `select` are generated for this action. For transient input sources, `inputsourceschange` events are still generated, but cancelling the beforexrselect event causes the action to be treated as an auxiliary action, similar to a secondary finger input.

§ 4. Initialization

The application MUST provide configuration for the DOM overlay through the `domOverlay` dictionary.

```
dictionary XRDOMOverlayInit {
  required Element root;
};
```

The ***root*** attribute specifies the ***overlay element*** that will be displayed to the user as the content of the DOM overlay. This is a required attribute, there is no default.

EXAMPLE 2

The following code requests DOM overlay as an optional feature.

```
let uiElement = document.getElementById('ui');
navigator.xr.requestSession('immersive-ar', {
    optionalFeatures: ['dom-overlay'],
    domOverlay: { root: uiElement } }).then((session) => {
    // session.domOverlayState.type is now set if supported,
    // or is null if the feature is not supported.
  }
}
```

While active, the DOM overlay element is automatically resized to fill the dimensions of the UA-provided DOM overlay rectangle. Its background color is automatically styled as transparent for the duration of the session.

NOTE: A UA MAY use the *Fullscreen API* § 5.2 User-agent level style sheet defaults to style the DOM overlay element, with an additional rule containing `background-color: rgba(0,0,0,0) !important;` to set the background transparent.

Once the session is active, the `domOverlayState` attribute provides information about the DOM overlay.

```
enum XRDOMOverlayType {
  "screen",
  "floating",
  "head-locked"
};

dictionary XRDOMOverlayState {
  XRDOMOverlayType type;
};
```

The user agent MUST set the `type` to indicate how the DOM overlay is being displayed. The value MUST remain unchanged for the duration of the session.

- An overlay type of ***screen*** indicates that the DOM overlay element covers the entire physical screen for a screen-based device, for example handheld AR. Its visual extent MUST match the visual extent of the `XRViewport`(s) used for `XRWebGLLayer` rendering. For a monoscopic display, this is a single viewport.

A steroscopic display screen would provide two viewports, in that case the DOM overlay MUST be rendered at the Z position matching the physical screen location, appearing identically in both eye views.

- An overlay type of ***`floating`*** indicates that the DOM overlay appears as a floating rectangle in space. The initial location of this rectangle in world space is up to the UA, and the UA MAY move it during the session to keep it in view, or support user-initiated manual placement.
- An overlay type of ***`head-locked`*** indicates that the DOM overlay follows the user's head movement consistently, appearing similar to a helmet heads-up display.

NOTE: From the user's point of view, a "floating" overlay is perceived as stationary when rendered as if anchored to a real-world location, and this style is a common choice for interactive display surfaces in VR. A "head-locked" overlay moves along with head rotations and does not have a fixed real-world location.

NOTE: Future versions of this spec may add additional attributes to the overlay state, for example the current location in world space for a floating overlay.

§ 5. Event handling for cross-origin content

The user agent MUST NOT provide poses or gamepad input state for user interactions with cross-origin content such as an `HTMLIFrameElement` nested within the DOM overlay element.

A user agent MAY meet this requirement by preventing user interactions with cross-origin content, for example by blocking DOM events that would normally be received by that content, or by not loading and displaying cross-origin content at all.

If the user agent supports interactions with cross-origin content in the DOM overlay, and if an input source's `targetRaySpace` intersects cross-origin content as the topmost event target, the UA MUST enable *WebXR Device API* § 13.4.3 Limiting data adjustment, and populate the pose for `XRSpace`s associated with that input source accordingly treating the `limit` boolean as true. In addition, the UA MUST NOT update gamepad data for this input source while poses are limited.

NOTE: The application does not receive pose updates for the controller or its targeting ray while poses are limited in this way. The UA is responsible for drawing a pointer ray or other appropriate visualization as needed to enable interactions.

NOTE: the restriction on updating gamepad data is intended to avoid information leakage from interactions with the cross-origin content to the application. For example, if an input source's primary action uses an analog trigger where the primary action happens at a certain trigger threshold, the application could infer when the user started and ended a primary action from the trigger value even if the corresponding events are blocked. Another example would be a device that uses a trackpad or joystick for text input in DOM content, where reading the axis values would allow the application to infer what text was being entered.

If a primary action ends inside cross-origin content, the UA MUST treat the primary action as cancelled, and MUST NOT send a `select` event. The UA MUST send the `selectend` event using the last available pose before entering the cross-origin content due to treating poses as limited.

If the input is a transient input source, and if the transient action begins inside cross-origin content, the user agent MUST delay adding the input source until the input location moves out of the cross-origin content. If the transient action ends while still inside the cross-origin content, the transient input source does not get added at all.

NOTE: On a handheld AR device using `screen`-mode input, this means that touches that stay inside cross-origin content don't create an input source or associated XR input events. If a drag movement starts inside cross-origin content, the input source is created at the location where the touch location leaves the cross-origin content, emitting a cancelable beforexrselect event as usual.

6. Security, Privacy, and Comfort Considerations

6.1. Protected functionality

The DOM overlay does not in itself introduce any new sensitive information. However, since it combines existing technologies, it's important to ensure that this combination does not lead to any unexpected interactions.

A primary design goal for this module was that the DOM overlay should follow existing semantics for 2D content where possible. Specifically, the information flows related to cross-origin embedded content should be similar to using iframes on a 2D page. For example, a 2D page can embed cross-origin content in an iframe, and then cover this iframe with a transparent element. In that case, the page will continue to receive mouse movement information, but the cross-origin content does not receive any input events in the covered area. For a DOM overlay, XR input event data is treated as similar to mouse movement data. Poses remain available to the outer page if there is no cross-origin content, or if the cross-origin content is not receiving input, but are limited (blocked) when interacting with cross-origin content.

Cross-origin content is potentially vulnerable to clickjacking threats. The UA MUST continue to apply mitigations such as *Content Security Policy 3* § 6.1.5 frame-src when iframes are used in a DOM overlay. The UA MAY implement additional restrictions specifically for cross-origin content in a DOM overlay if necessary to address specific threats.

§ Changes

§ Changes from the First Public Working Draft 31 August 2021

§ 7. Acknowledgements

The following individuals have contributed to the design of the WebXR DOM Overlay specification:

- Brandon Jones (Google)
- Nell Waliczek (Amazon [Microsoft until 2018])

§ Conformance

§ Document conventions

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for

readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

EXAMPLE 3

This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

Note, this is an informative note.

Conformant Algorithms

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("must", "should", "may", etc) used in introducing the algorithm.

Conformance requirements phrased as algorithms or specific steps can be implemented in any manner, so long as the end result is equivalent. In particular, the algorithms defined in this specification are intended to be easy to understand and are not intended to be performant. Implementers are encouraged to optimize.

Index

Terms defined by this specification

§ Terms defined by reference

[CSS-POSITION-4] defines the following terms:

- top layer

[CSS21] defines the following terms:

- stacking context

[DOM] defines the following terms:

- Element
- fire an event
- preventDefault()
- target

[FILTER-EFFECTS-2] defines the following terms:

- backdrop root

[FULLSCREEN] defines the following terms:

- requestFullscreen()

[HTML] defines the following terms:

- EventHandler
- GlobalEventHandlers
- HTMLIFrameElement
- Window
- queue a task
- rendering opportunity
- requestAnimationFrame(callback)

[SELECTORS-4] defines the following terms:

- pseudo-class

[UIEVENTS] defines the following terms:

- topmost event target

[WEBIDL] defines the following terms:

- NotSupportedError

[WEBXR] defines the following terms:

- XRInputSource
- XRSession
- XRSessionEvent
- XRSessionInit
- XRSpace
- XRViewport
- XRWebGLLayer
- inputsourceschange
- optionalFeatures
- requestAnimationFrame(callback)
- requestSession(mode)
- requiredFeatures
- selectend
- selectstart
- targetRaySpace

[WEBXR-DOM-OVERLAYS-1] defines the following terms:

- :xr-overlay

References

Normative References

[CSP3]
Mike West; Antonio Sartori. *Content Security Policy Level 3*. 9 September 2024. WD. URL: https://www.w3.org/TR/CSP3/

[CSS-POSITION-4]
CSS Positioned Layout Module Level 4. Editor's Draft. URL: https://drafts.csswg.org/css-position-4/

[CSS21]
Bert Bos; et al. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. 7 June 2011. REC. URL: https://www.w3.org/TR/CSS21/

[DOM]
Anne van Kesteren. *DOM Standard*. Living Standard. URL: https://dom.spec.whatwg.org/

[FILTER-EFFECTS-2]
Filter Effects Module Level 2. Editor's Draft. URL: https://drafts.fxtf.org/filter-effects-2/

[FULLSCREEN]
Philip Jägenstedt. *Fullscreen API Standard*. Living Standard. URL: https://fullscreen.spec.whatwg.org/

[HTML]
Anne van Kesteren; et al. *HTML Standard*. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://datatracker.ietf.org/doc/html/rfc2119

[SELECTORS-4]
Elika Etemad; Tab Atkins Jr.. *Selectors Level 4*. 11 November 2022. WD. URL: https://www.w3.org/TR/selectors-4/

[UIEVENTS]
Gary Kacmarcik; Travis Leithead. *UI Events*. 7 September 2024. WD. URL: https://www.w3.org/TR/uievents/

[WEBIDL]
Edgar Chen; Timothy Gu. *Web IDL Standard*. Living Standard. URL: https://webidl.spec.whatwg.org/

[WEBXR]
Brandon Jones; Manish Goregaokar; Rik Cabanier. *WebXR Device API*. 21 August 2024. CR. URL: https://www.w3.org/TR/webxr/

[WEBXR-DOM-OVERLAYS-1]
Piotr Bialecki. *WebXR DOM Overlays Module*. 17 November 2022. WD. URL: https://www.w3.org/TR/webxr-dom-overlays-1/

§ IDL Index

```
partial interface mixin GlobalEventHandlers {
  attribute EventHandler onbeforexrselect;
};

partial dictionary XRSessionInit {
  XRDOMOverlayInit? domOverlay;
};

partial interface XRSession {
  readonly attribute XRDOMOverlayState? domOverlayState;
};

dictionary XRDOMOverlayInit {
  required Element root;
};

enum XRDOMOverlayType {
  "screen",
  "floating",
  "head-locked"
};

dictionary XRDOMOverlayState {
  XRDOMOverlayType type;
};
```