MDN Web Docs moz://a

Window

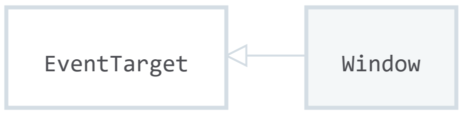
The **`Window`** interface represents a window containing a DOM document; the `document` property points to the DOM document loaded in that window.

A window for a given document can be obtained using the `document.defaultView` property.

A global variable, `window`, representing the window in which the script is running, is exposed to JavaScript code.

The `Window` interface is home to a variety of functions, namespaces, objects, and constructors which are not necessarily directly associated with the concept of a user interface window. However, the `Window` interface is a suitable place to include these items that need to be globally available. Many of these are documented in the JavaScript Reference and the DOM Reference.

In a tabbed browser, each tab is represented by its own `Window` object; the global `window` seen by JavaScript code running within a given tab always represents the tab in which the code is running. That said, even in a tabbed browser, some properties and methods still apply to the overall window that contains the tab, such as `resizeTo()` and `innerHeight`. Generally, anything that can't reasonably pertain to a tab pertains to the window instead.



Constructors

See also the DOM Interfaces.

`DOMParser`

`DOMParser` can parse XML or HTML source stored in a string into a DOM Document. `DOMParser` is specified in DOM Parsing and Serialization .

`HTMLImageElement.Image`

Used for creating an `HTMLImageElement`.

`HTMLOptionElement.Option`

Used for creating an `HTMLOptionElement`.

`StaticRange` 🧪 **Read only**

Returns a `StaticRange()` constructor which creates a `StaticRange` object.

`Worker`

Used for creating a Web worker.

`XMLSerializer`

Converts a DOM tree into XML or HTML source.

Properties

This interface inherits properties from the `EventTarget` interface and implements properties from the `WindowOrWorkerGlobalScope` and `WindowEventHandlers` mixins.

Note that properties which are objects (e.g.,. for overriding the prototype of built-in elements) are listed in a separate section below.

`Window.clientInformation` **Read only**

An alias for `Window.navigator`.

`Window.closed` **Read only**

This property indicates whether the current window is closed or not.

`Window.console` **Read only**

Returns a reference to the console object which provides access to the browser's debugging console.

`Window.customElements` **Read only**

Returns a reference to the `CustomElementRegistry` object, which can be used to register new custom elements and get information about previously registered custom elements.

`Window.crypto` **Read only**

Returns the browser crypto object.

`Window.devicePixelRatio` **Read only**

Returns the ratio between physical pixels and device independent pixels in the current display.

`Window.document` **Read only**

Returns a reference to the document that the window contains.

`Window.DOMMatrix` **Read only** ⚗

Returns a reference to a `DOMMatrix` object, which represents 4x4 matrices, suitable for 2D and 3D operations.

`Window.DOMMatrixReadOnly` **Read only** ⚗

Returns a reference to a `DOMMatrixReadOnly` object, which represents 4x4 matrices, suitable for 2D and 3D operations.

`Window.DOMPoint` **Read only** ⚗

Returns a reference to a `DOMPoint` object, which represents a 2D or 3D point in a coordinate system.

`Window.DOMPointReadOnly` **Read only** ⚗

Returns a reference to a `DOMPointReadOnly` object, which represents a 2D or 3D point in a coordinate system.

`Window.DOMQuad` **Read only** ⚗

Returns a reference to a `DOMQuad` object, which provides represents a quadrilaterial object, that is one having four corners and four sides.

`Window.DOMRect` **Read only** ⚗

Returns a reference to a `DOMRect` object, which represents a rectangle.

`Window.DOMRectReadOnly` **Read only** ⚗

Returns a reference to a `DOMRectReadOnly` object, which represents a rectangle.

`Window.event` 🗑 **Read only**

Returns the **current event**, which is the event currently being handled by the JavaScript code's context, or `undefined` if no event is currently being handled. The `Event` object passed directly to event handlers should be used instead whenever possible.

`Window.frameElement` **Read only**

Returns the element in which the window is embedded, or null if the window is not embedded.

`Window.frames` **Read only**

Returns an array of the subframes in the current window.

`Window.fullScreen`

This property indicates whether the window is displayed in full screen or not.

`Window.history` **Read only**

Returns a reference to the history object.

`Window.innerHeight` **Read only**

Gets the height of the content area of the browser window including, if rendered, the horizontal scrollbar.

`Window.innerWidth` **Read only**

Gets the width of the content area of the browser window including, if rendered, the vertical scrollbar.

`isSecureContext` ⚗ **Read only**

Indicates whether a context is capable of using features that require secure contexts.

`Window.length` **Read only**

Returns the number of frames in the window. See also `window.frames`.

`Window.location`

Gets/sets the location, or current URL, of the window object.

`Window.locationbar` **Read only**

Returns the locationbar object, whose visibility can be toggled in the window.

`Window.localStorage` **Read only**

Returns a reference to the local storage object used to store data that may only be accessed by the origin that created it.

`Window.menubar` **Read only**

Returns the menubar object, whose visibility can be toggled in the window.

`Window.messageManager` 👎

Returns the message manager object for this window.

`Window.mozInnerScreenX` **Read only** 👎

Returns the horizontal (X) coordinate of the top-left corner of the window's viewport, in screen coordinates. This value is reported in CSS pixels. See `mozScreenPixelsPerCSSPixel` in `nsIDOMWindowUtils` for a conversion factor to adapt to screen pixels if needed.

`Window.mozInnerScreenY` **Read only** 👎

Returns the vertical (Y) coordinate of the top-left corner of the window's viewport, in screen coordinates. This value is reported in CSS pixels. See `mozScreenPixelsPerCSSPixel` for a conversion factor to adapt to screen pixels if needed.

`Window.name`

Gets/sets the name of the window.

`Window.navigator` **Read only**

Returns a reference to the navigator object.

`Window.opener`

Returns a reference to the window that opened this current window.

`Window.outerHeight` **Read only**

Gets the height of the outside of the browser window.

`Window.outerWidth` **Read only**

Gets the width of the outside of the browser window.

`Window.pageXOffset` **Read only**

An alias for `window.scrollX`.

`Window.pageYOffset` **Read only**

An alias for `window.scrollY`

`Window.parent` **Read only**

Returns a reference to the parent of the current window or subframe.

`Window.performance` **Read only**

Returns a `Performance` object, which includes the `timing` and `navigation` attributes, each of which is an object providing performance-related data. See also Using Navigation Timing for additional information and examples.

`Window.personalbar` **Read only**

Returns the personalbar object, whose visibility can be toggled in the window.

`Window.screen` **Read only**

Returns a reference to the screen object associated with the window.

`Window.screenX` and `Window.screenLeft` **Read only**

Both properties return the horizontal distance from the left border of the user's browser viewport to the left side of the screen.

`Window.screenY` and `Window.screenTop` **Read only**

Both properties return the vertical distance from the top border of the user's browser viewport to the top side of the screen.

`Window.scrollbars` **Read only**

Returns the scrollbars object, whose visibility can be toggled in the window.

`Window.scrollMaxX` 👎 **Read only**

The maximum offset that the window can be scrolled to horizontally, that is the document width minus the viewport width.

`Window.scrollMaxY` 👎 **Read only**

The maximum offset that the window can be scrolled to vertically (i.e., the document height minus the viewport height).

`Window.scrollX` **Read only**

Returns the number of pixels that the document has already been scrolled horizontally.

`Window.scrollY` **Read only**

Returns the number of pixels that the document has already been scrolled vertically.

`Window.self` **Read only**

Returns an object reference to the window object itself.

`Window.sessionStorage`

Returns a reference to the session storage object used to store data that may only be accessed by the origin that created it.

`Window.sidebar` 🗑 👎 **Read only**

Returns a reference to the window object of the sidebar.

`Window.speechSynthesis` **Read only**

Returns a `SpeechSynthesis` object, which is the entry point into using Web Speech API speech synthesis functionality.

`Window.status` 🗑

Gets/sets the text in the statusbar at the bottom of the browser.

`Window.statusbar` **Read only**

Returns the statusbar object, whose visibility can be toggled in the window.

`Window.toolbar` **Read only**

Returns the toolbar object, whose visibility can be toggled in the window.

`Window.top` **Read only**

Returns a reference to the topmost window in the window hierarchy. This property is read only.

`Window.visualViewport` **Read only**

Returns a `VisualViewport` object which represents the visual viewport for a given window.

`Window.window` **Read only**

Returns a reference to the current window.

`window[0]`, `window[1]`, etc.

Returns a reference to the `window` object in the frames. See `Window.frames` for more details.

Properties implemented from elsewhere

`caches` **Read only**

Returns the `CacheStorage` object associated with the current context. This object enables functionality such as storing assets for offline use, and generating custom responses to requests.

`indexedDB` **Read only**

Provides a mechanism for applications to asynchronously access capabilities of indexed databases; returns an `IDBFactory` object.

`isSecureContext` **Read only**

Returns a boolean indicating whether the current context is secure (`true`) or not (`false`).

`origin` **Read only**

Returns the global object's origin, serialized as a string. (This does not yet appear to be implemented in any browser.)

Deprecated properties

`Window.content` **and `Window._content`** 👎 🗑 **Read only**

Returns a reference to the content element in the current window. Since Firefox 57 (initially

Nightly-only), both versions are only available from chrome (privileged) code, and not available to the web anymore.

`Window.defaultStatus` 🗑

Gets/sets the status bar text for the given window.

`Window.dialogArguments` 🗑 **Read only**

Gets the arguments passed to the window (if it's a dialog box) at the time `window.showModalDialog()` was called. This is an `nsIArray`.

`Window.mozPaintCount` 👎 🗑

Returns the number of times the current document has been rendered to the screen in this window. This can be used to compute rendering performance.

`Window.orientation` **Read only** 🗑

Returns the orientation in degrees (in 90 degree increments) of the viewport relative to the device's natural orientation.

`Window.returnValue` 🗑

The return value to be returned to the function that called `window.showModalDialog()` to display the window as a modal dialog.

Methods

This interface inherits methods from the `EventTarget` interface and implements methods from `WindowOrWorkerGlobalScope` and `EventTarget`.

`Window.alert()`

Displays an alert dialog.

`Window.blur()`

Sets focus away from the window.

`Window.cancelAnimationFrame()` ⚗

Enables you to cancel a callback previously scheduled with `Window.requestAnimationFrame`.

`Window.cancelIdleCallback()` ⚗

Enables you to cancel a callback previously scheduled with `Window.requestIdleCallback`.

`Window.clearImmediate()`

Cancels the repeated execution set using `setImmediate`.

`Window.close()`

Closes the current window.

`Window.confirm()`

Displays a dialog with a message that the user needs to respond to.

`Window.dump()` 👎

Writes a message to the console.

`Window.find()`

Searches for a given string in a window.

`Window.focus()`

Sets focus on the current window.

`Window.getComputedStyle()`

Gets computed style for the specified element. Computed style indicates the computed values of all CSS properties of the element.

`Window.getDefaultComputedStyle()` 👎

Gets default computed style for the specified element, ignoring author stylesheets.

`Window.getSelection()`

Returns the selection object representing the selected item(s).

`Window.matchMedia()`

Returns a `MediaQueryList` object representing the specified media query string.

`Window.moveBy()`

Moves the current window by a specified amount.

`Window.moveTo()`

Moves the window to the specified coordinates.

`Window.open()`

Opens a new window.

`Window.postMessage()`

Provides a secure means for one window to send a string of data to another window, which need not be within the same domain as the first.

`Window.print()`

Opens the Print Dialog to print the current document.

`Window.prompt()`

Returns the text entered by the user in a prompt dialog.

`Window.requestAnimationFrame()`

Tells the browser that an animation is in progress, requesting that the browser schedule a repaint of the window for the next animation frame.

`Window.requestIdleCallback()` ⚗

Enables the scheduling of tasks during a browser's idle periods.

`Window.resizeBy()`

Resizes the current window by a certain amount.

`Window.resizeTo()`

Dynamically resizes window.

`Window.scroll()`

Scrolls the window to a particular place in the document.

`Window.scrollBy()`

Scrolls the document in the window by the given amount.

`Window.scrollByLines()` 👎

Scrolls the document by the given number of lines.

`Window.scrollByPages()` 👎

Scrolls the current document by the specified number of pages.

`Window.scrollTo()`

Scrolls to a particular set of coordinates in the document.

`Window.setImmediate()`

Executes a function after the browser has finished other heavy tasks

`Window.setResizable()` 👎

Toggles a user's ability to resize a window.

`Window.sizeToContent()` 👎

Sizes the window according to its content.

`Window.showOpenFilePicker()`

Shows a file picker that allows a user to select a file or multiple files.

`Window.showSaveFilePicker()`

Shows a file picker that allows a user to save a file.

`Window.showDirectoryPicker()`

Displays a directory picker which allows the user to select a directory.

`Window.stop()`

This method stops window loading.

`Window.updateCommands()` 👎

Updates the state of commands of the current chrome window (UI).

Methods implemented from elsewhere

`EventTarget.addEventListener()`

Register an event handler to a specific event type on the window.

Register an event handler to a specific event type on the window.

`EventTarget.dispatchEvent()`

Used to trigger an event.

`atob()`

Decodes a string of data which has been encoded using base-64 encoding.

`btoa()`

Creates a base-64 encoded ASCII string from a string of binary data.

`clearInterval()`

Cancels the repeated execution set using `setInterval()`.

`clearTimeout()`

Cancels the delayed execution set using `setTimeout()`.

`createImageBitmap()`

Accepts a variety of different image sources, and returns a `Promise` which resolves to an `ImageBitmap`. Optionally the source is cropped to the rectangle of pixels originating at *(sx, sy)* with width sw, and height sh.

`fetch()`

Starts the process of fetching a resource from the network.

`EventTarget.removeEventListener`

Removes an event listener from the window.

`setInterval()`

Schedules a function to execute every time a given number of milliseconds elapses.

`setTimeout()`

Schedules a function to execute in a given amount of time.

`reportError()`

Reports an error in a script, emulating an unhandled exception.

Deprecated methods

`Window.back()`

Moves back one in the window history. This method is deprecated; you should instead use `window.history.back()`.

`Window.captureEvents()`

Registers the window to capture all events of the specified type.

`Window.forward()`

Moves the window one document forward in the history. This method is deprecated; you should instead use `window.history.forward()`.

`Window.home()`

Returns the browser to the home page.

`Window.openDialog()`

Opens a new dialog window.

`Window.releaseEvents()`

Releases the window from trapping events of a specific type.

`Window.showModalDialog()`

Displays a modal dialog.

Event handlers

These are properties of the window object that can be set to establish event handlers for the various things that can happen in the window that might be of interest.

This interface inherits event handlers from the `EventTarget` interface and implements event handlers from `WindowEventHandlers`.

`Window.onappinstalled`

Called when the page is installed as a webapp. See `appinstalled` event.

`Window.onbeforeinstallprompt` 👎

An event handler property dispatched before a user is prompted to save a web site to a home screen on mobile.

`Window.ondevicemotion`

Called if accelerometer detects a change (For mobile devices)

`Window.ondeviceorientation`

Called when the orientation is changed (For mobile devices)

`Window.ondeviceorientationabsolute` 👎

An event handler property for any device orientation changes.

`Window.ondeviceproximity` 🗑

An event handler property for device proximity event (see `DeviceProximityEvent`)

`Window.ongamepadconnected`

Represents an event handler that will run when a gamepad is connected (when the `gamepadconnected` event fires).

`Window.ongamepaddisconnected`

Represents an event handler that will run when a gamepad is disconnected (when the `gamepaddisconnected` event fires).

`WindowEventHandlers.onrejectionhandled`

An event handler for handled `Promise` rejection events.

`Window.onuserproximity` 🗑

An event handler property for user proximity events (see `UserProximityEvent`).

`Window.onvrdisplayconnect` 🗑

Represents an event handler that will run when a compatible VR device has been connected to the computer (when the `vrdisplayconnected` event fires).

`Window.onvrdisplaydisconnect` 🗑

Represents an event handler that will run when a compatible VR device has been disconnected from the computer (when the `vrdisplaydisconnected` event fires).

`Window.onvrdisplayactivate` 🗑

Represents an event handler that will run when a display is able to be presented to (when the `vrdisplayactivate` event fires), for example if an HMD has been moved to bring it out of standby, or woken up by being put on.

`Window.onvrdisplaydeactivate` 🗑

Represents an event handler that will run when a display can no longer be presented to (when the `vrdisplaydeactivate` event fires), for example if an HMD has gone into standby or sleep mode due to a period of inactivity.

`Window.onvrdisplayblur` 🗑

Represents an event handler that will run when presentation to a display has been paused for some reason by the browser, OS, or VR hardware (when the `vrdisplayblur` event fires) — for example, while the user is interacting with a system menu or browser, to prevent tracking or loss of experience.

`Window.onvrdisplayfocus` 🗑

Represents an event handler that will run when presentation to a display has resumed after being blurred (when the `vrdisplayfocus` event fires).

`Window.onvrdisplaypresentchange` 🗑

represents an event handler that will run when the presenting state of a VR device changes — i.e. goes from presenting to not presenting, or vice versa (when the `vrdisplaypresentchange` event fires).

Event handlers implemented from elsewhere

`GlobalEventHandlers.onabort`

Called when the loading of a resource has been aborted, such as by a user canceling the load while it is still in progress

`WindowEventHandlers.onafterprint`

Called when the print dialog box is closed. See `afterprint` event.

`WindowEventHandlers.onbeforeprint`

Called when the print dialog box is opened. See `beforeprint` event.

`WindowEventHandlers.onbeforeunload`

An event handler property for before-unload events on the window.

`GlobalEventHandlers.onblur`

Called after the window loses focus, such as due to a popup.

`GlobalEventHandlers.onchange`

An event handler property for change events on the window.

`GlobalEventHandlers.onclick`

Called after the ANY mouse button is pressed & released

`GlobalEventHandlers.ondblclick`

Called when a double click is made with ANY mouse button.

`GlobalEventHandlers.onclose`

Called after the window is closed

`GlobalEventHandlers.oncontextmenu`

Called when the RIGHT mouse button is pressed

`GlobalEventHandlers.onerror`

Called when a resource fails to load OR when an error occurs at runtime. See `error` event.

`GlobalEventHandlers.onfocus`

Called after the window receives or regains focus. See `focus` events.

`WindowEventHandlers.onhashchange`

An event handler property for `hashchange` events on the window; called when the part of the URL after the hash mark ("#") changes.

`GlobalEventHandlers.oninput`

Called when the value of an <input> element changes

`GlobalEventHandlers.onkeydown`

Called when you begin pressing ANY key. See `keydown` event.

`GlobalEventHandlers.onkeypress`

Called when a key (except Shift, Fn, and CapsLock) is in pressed position. See `keypress` event.

`GlobalEventHandlers.onkeyup`

Called when you finish releasing ANY key. See `keyup` event.

`WindowEventHandlers.onlanguagechange`

An event handler property for `languagechange` events on the window.

`GlobalEventHandlers.onload`

Called after all resources and the DOM are fully loaded. WILL NOT get called when the page is loaded from cache, such as with back button.

`WindowEventHandlers.onmessage`

Is an event handler representing the code to be called when the `message` event is raised.

`GlobalEventHandlers.onmousedown`

Called when ANY mouse button is pressed.

`GlobalEventHandlers.onmousemove`

Called continously when the mouse is moved inside the window.

`GlobalEventHandlers.onmouseout`

Called when the pointer leaves the window.

`GlobalEventHandlers.onmouseover`

Called when the pointer enters the window

`GlobalEventHandlers.onmouseup`

Called when ANY mouse button is released

`WindowEventHandlers.onoffline`

Called when network connection is lost. See `offline` event.

`WindowEventHandlers.ononline`

Called when network connection is established. See `online` event.

`WindowEventHandlers.onpagehide`

Called when the user navigates away from the page, before the onunload event. See `pagehide` event.

`WindowEventHandlers.onpageshow`

Called after all resources and the DOM are fully loaded. See `pageshow` event.

`WindowEventHandlers.onpopstate`

Called when a back button is pressed.

`GlobalEventHandlers.onreset`

Called when a form is reset

`GlobalEventHandlers.onresize`

Called continuously as you are resizing the window.

`GlobalEventHandlers.onscroll`

Called when the scroll bar is moved via ANY means. If the resource fully fits in the window, then this event cannot be invoked

`GlobalEventHandlers.onwheel`

Called when the mouse wheel is rotated around any axis

`GlobalEventHandlers.onselect`

Called after text in an input field is selected

`GlobalEventHandlers.onselectionchange`

Is an event handler representing the code to be called when the `selectionchange` event is raised.

`WindowEventHandlers.onstorage`

Called when there is a change in session storage or local storage. See `storage` event

`GlobalEventHandlers.onsubmit`

Called when a form is submitted

`WindowEventHandlers.onunhandledrejection`

An event handler for unhandled `Promise` rejection events.

`WindowEventHandlers.onunload`

Called when the user navigates away from the page.

Events

Listen to these events using `addEventListener()` or by assigning an event listener to the `oneventname` property of this interface.

`error`

Fired when a resource failed to load, or can't be used. For example, if a script has an execution error or an image can't be found or is invalid. Also available via the `onerror` property.

`languagechange`

Fired at the global scope object when the user's preferred language changes. Also available via the `onlanguagechange` property.

`orientationchange`

Fired when the orientation of the device has changed. Also available via the `onorientationchange` property.

`devicemotion`

Fired at a regular interval, indicating the amount of physical force of acceleration the device is receiving and the rate of rotation, if available.

`deviceorientation`

Fired when fresh data is available from the magnetometer orientation sensor about the current orientation of the device as compared to the Earth coordinate frame.

`resize`

Fired when the window has been resized. Also available via the `onresize` property.

`storage`

Fired when a storage area (`localStorage` or `sessionStorage`) has been modified in the context of another document. Also available via the `onstorage` property.

Animation events

`animationcancel`

Fired when an animation unexpectedly aborts. Also available via the `onanimationcancel` property.

`animationend`

Fired when an animation has completed normally. Also available via the `onanimationend` property.

`animationiteration`

Fired when an animation iteration has completed. Also available via the `onanimationiteration` property.

`animationstart`

Fired when an animation starts. Also available via the `onanimationstart` property.

Clipboard events

`copy`

Fired when the user initiates a copy action through the browser's user interface. Also available via the `oncopy` property.

`cut`

Fired when the user initiates a cut action through the browser's user interface. Also available via the `oncut` property.

`paste`

Fired when the user initiates a paste action through the browser's user interface. Also available via the `onpaste` property.

Connection events

`offline`

Fired when the browser has lost access to the network and the value of `navigator.onLine` has switched to `false`. Also available via the `onoffline` property.

`online`

Fired when the browser has gained access to the network and the value of `navigator.onLine` has switched to `true`. Also available via the `ononline` property.

Focus events

`blur`

Fired when an element has lost focus. Also available via the `onblur` property.

`focus`

Fired when an element has gained focus. Also available via the `onfocus` property

Gamepad events

`gamepadconnected`

Fired when the browser detects that a gamepad has been connected or the first time a button/axis of the gamepad is used. Also available via the `ongamepadconnected` property.

`gamepaddisconnected`

Fired when the browser detects that a gamepad has been disconnected. Also available via the `ongamepaddisconnected` property

History events

`hashchange`

Fired when the fragment identifier of the URL has changed (the part of the URL beginning with and following the `#` symbol). Also available via the `onhashchange` property.

`pagehide`

Sent when the browser hides the current document while in the process of switching to displaying in its place a different document from the session's history. This happens, for example, when the user clicks the Back button or when they click the Forward button to move ahead in session history. Also available through the `onpagehide` event handler property.

`pageshow`

Sent when the browser makes the document visible due to navigation tasks, including not only when the page is first loaded, but also situations such as the user navigating back to the page after having navigated to another within the same tab. Also available using the `onpageshow` event handler property.

`popstate`

Fired when the active history entry changes. Also available using the `onpopstate` event handler property.

Load & unload events

`beforeunload`

Fired when the window, the document and its resources are about to be unloaded. Also available via the `onbeforeunload` property.

`DOMContentLoaded`

Fired when the document has been completely loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading.

`load`

Fired when the whole page has loaded, including all dependent resources such as stylesheets images. Also available via the `onload` property.

`unload`

Fired when the document or a child resource is being unloaded. Also available via the `onunload` property.

Manifest events

`appinstalled`

Fired when the browser has successfully installed a page as an application. Also available via the `onappinstalled` property.

`beforeinstallprompt`

Fired when a user is about to be prompted to install a web application. Also available via the `onbeforeinstallprompt` property.

Messaging events

`message`

Fired when the window receives a message, for example from a call to `Window.postMessage()` from another browsing context. Also available via the `onmessage` property.

`messageerror`

Fired when a `Window` object receives a message that can't be deserialized. Also available via the `onmessageerror` property.

Print events

`afterprint`

Fired after the associated document has started printing or the print preview has been closed. Also available via the `onafterprint` property.

`beforeprint`

Fired when the associated document is about to be printed or previewed for printing. Also available via the `onbeforeprint` property.

Promise rejection events

`rejectionhandled`

Sent every time a JavaScript `Promise` is rejected, regardless of whether or not there is a handler in place to catch the rejection. Also available through the `onrejectionhandled` event handler property.

unhandledrejection

Sent when a JavaScript `Promise` is rejected but there is no handler in place to catch the rejection. Also available using the `onunhandledrejection` event handler property.

Transition events

transitioncancel

Fired when a CSS transition is canceled. Also available via the `ontransitioncancel` property.

transitionend

Fired when a CSS transition has completed. Also available via the `ontransitionend` property.

transitionrun

Fired when a CSS transition is first created. Also available via the `ontransitionrun` property.

transitionstart

Fired when a CSS transition has actually started. Also available via the `ontransitionstart` property.

WebVR events

vrdisplayactivate

Fired when a VR display becomes available to be presented to, for example if an HMD has been moved to bring it out of standby, or woken up by being put on. Also available via the `onvrdisplayactivate` property.

vrdisplayblur

Fired when presentation to a VR display has been paused for some reason by the browser, OS, or VR hardware. Also available via the `onvrdisplayblur` property.

vrdisplayconnect

Fired when a compatible VR display is connected to the computer. Also available via the `onvrdisplayconnect` property.

`vrdisplaydeactivate`

Fired when a VR display can no longer be presented to, for example if an HMD has gone into standby or sleep mode due to a period of inactivity. Also available via the `onvrdisplaydeactivate` property.

`vrdisplaydisconnect`

Fired when a compatible VR display is disconnected from the computer. Also available via the `onvrdisplaydisconnect` property.

`vrdisplayfocus`

Fired when presentation to a VR display has resumed after being blurred. Also available via the `onvrdisplayfocus` property.

`vrdisplaypresentchange`

Fired when the presenting state of a VR display changes — i.e. goes from presenting to not presenting, or vice versa. Also available via the `onvrdisplaypresentchange` property.

`vrdisplaypointerrestricted`

Fired when the VR display's pointer input is restricted to consumption via a pointerlocked element. Also available via the `onvrdisplaypointerrestricted` property.

`vrdisplaypointerunrestricted`

Fired when the VR display's pointer input is no longer restricted to consumption via a pointerlocked element. Also available via the `onvrdisplaypointerunrestricted` property.

Interfaces

See DOM Reference.

Specifications

Specification