Window: open() method

- Syntax
  - Parameters
  - Return value
- Description
- Examples
  - Opening a new tab
  - Opening a popup
- Progressive enhancement
  - Provide alternative ways when JavaScript is disabled
    - HTML
    - JavaScript
  - Reuse existing windows and avoid target="_blank"
    - HTML
    - JavaScript
- Same-origin policy
- Accessibility concerns
  - Avoid resorting to window.open()
  - Never use window.open() inline in HTML
  - Always identify links leading to a secondary window
- Specifications
- Browser compatibility
- See also

The `open()` method of the `Window` interface loads a specified resource into a new or existing browsing context (that is, a tab, a window, or an iframe) under a specified name.

Syntax

```
open()
open(url)
open(url, target)
open(url, target, windowFeatures)
```

Parameters

- `url`
  - : A string indicating the URL or path of the resource to be loaded. If an empty string (`""`) is specified or this parameter is omitted, a blank page is opened into the targeted browsing context.
- `target`
  - : A string, without whitespace, specifying the name of the browsing context the resource is being loaded into. If the name doesn't identify an existing context, a new context is created and given the specified name. The special `target` keywords, `_self`, `_blank` (default), `_parent`, `_top`, and `_unfencedTop` can also be used.
    `_unfencedTop` is only relevant to fenced frames.
    This name can be used as the `target` attribute of `<a>` or `<form>` elements.
- `windowFeatures`
  - : A string containing a comma-separated list of window features in the form `name=value`. Boolean values can be set to true using one of: `name`, `name=yes`, `name=true`, or `name=n` where `n` is any non-zero integer. These features include options such as the window's default size and position, whether or not to open a minimal popup window, and so forth. The following options are supported:
    - `attributionsrc`
      - : Indicates that you want the browser to send an Attribution-Reporting-Eligible header along with the `open()` call. This call must be made with transient activation (i.e., inside a user interaction event handle such as `click`), within five seconds of user interaction. On the server-side this is used to trigger sending an Attribution-Reporting-Register-Source header in the response to complete registration of an attribution source.
        In addition, the browser is also triggered to store the associated source data (as provided in the Attribution-Reporting-Register-Source response header) when the `open()` method completes.
        See the Attribution Reporting API for more details.

        > **NOTE:** `open()` calls cannot be used to register an attribution trigger.

    - `popup`

- : By default, `window.open` opens the page in a new tab. If `popup` is set to true, it requests that a minimal popup window be used. The UI features included in the popup window will be automatically decided by the browser, generally including an address bar only. If `popup` is present and set to false, a new tab is still opened.
  There are a few legacy features, which used to control UI features of the opened window. In modern browsers, they only have the effect of requesting a popup. If `popup` is unspecified, and `windowFeatures` contains any features (including unrecognized ones) other than `noopener`, `noreferrer`, or `attributionsrc`, the window is also opened as a popup if any of the following conditions apply:
  - `location` and `toolbar` are both false or absent
  - `menubar` is false or absent
  - `resizable` is false
  - `scrollbars` is false or absent
  - `status` is false or absent

  Otherwise the window is opened as a tab.
- `width` or `innerWidth`
  - : Specifies the width of the content area, including scrollbars. The minimum required value is 100.
- `height` or `innerHeight`
  - : Specifies the height of the content area, including scrollbars. The minimum required value is 100.
- `left` or `screenX`
  - : Specifies the distance in pixels from the left side of the work area as defined by the user's operating system where the new window will be generated.
- `top` or `screenY`
  - : Specifies the distance in pixels from the top side of the work area as defined by the user's operating system where the new window will be generated.
- `noopener`
  - : If this feature is set, the new window will not have access to the originating window via `Window.opener` and returns `null`.
    When `noopener` is used, non-empty target names, other than `_top`, `_self`, and `_parent`, are treated like `_blank` in terms of deciding whether to open a new browsing context.
- `noreferrer`
  - : If this feature is set, the browser will omit the `Referer` header, as well as set `noopener` to true. See `rel="noreferrer"` for more information.

A `null` value is treated the same as the empty string (`""`).

> **NOTE:** Requested position (`top`, `left`), and requested dimension (`width`, `height`) values in `windowFeatures` **will be corrected** if any of such requested value does not allow the entire browser popup to be rendered within the work area for applications of the user's operating system. In other words, no part of the new popup can be initially positioned offscreen.

Return value

If the browser successfully opens the new browsing context, a `WindowProxy` object is returned. The returned reference can be used to access properties and methods of the new context as long as it complies with the same-origin policy security requirements.

If the Cross-Origin-Opener-Policy HTTP header is being used, and the document policies are such that the document is opened in a new browsing context group, references to the opened window are severed and the returned object will indicate that the opened window is closed (closed is `true`).

`null` is returned if the browser fails to open the new browsing context, for example because it was blocked by a browser popup blocker.

Description

The `Window` interface's `open()` method takes a URL as a parameter, and loads the resource it identifies into a new or existing browsing context. The `target` parameter determines which window, tab, or frame, to load the resource into, and the `windowFeatures` parameter can be used to control the features of the new window, such as whether it is a tab or a popup with minimal UI features, its size and position, and so on.

When `window.open()` creates a new browsing context (i.e., when no existing window with that name is found), the window initially contains `about:blank`. If a different URL was provided, it is loaded asynchronously, and the global object is reused for that navigation if it is same-origin — so any properties set on the window before the load may persist. If target refers to an existing navigable (`_self`, `_parent`, `_top`, or a known window name), no `about:blank` phase occurs — the browser navigates the existing context directly.

Modern browsers have strict popup blocker policies. Popup windows must be opened in direct response to user input, and a separate user gesture event is required for each `Window.open()` call. This prevents sites from spamming users with lots of windows. However, this poses an issue for multi-window applications. To work around this limitation, you can design your applications to:

- Open no more than one new window at once.
- Reuse existing windows to display different pages.
- Advise users on how to update their browser settings to allow multiple windows.

Examples

Opening a new tab

```
window.open("https://www.mozilla.org/", "mozillaTab");
```

Opening a popup

Alternatively, the following example demonstrates how to open a popup, using the `popup` feature.

```
window.open("https://www.mozilla.org/", "mozillaWindow", "popup");
```

It is possible to control the size and position of the new popup:

```
const windowFeatures = "left=100,top=100,width=320,height=320";
const handle = window.open(
  "https://www.mozilla.org/",
  "mozillaWindow",
  windowFeatures,
);
if (!handle) {
  // The window wasn't allowed to open
  // This is likely caused by built-in popup blockers.
  // …
}
```

Progressive enhancement

In some cases, JavaScript is disabled or unavailable and `window.open()` will not work. Instead of solely relying on the presence of this feature, we can provide an alternative solution so that the site or application still functions.

Provide alternative ways when JavaScript is disabled

If JavaScript support is disabled or non-existent, then the user agent will create a secondary window accordingly or will render the referenced resource according to its handling of the `target` attribute. The goal and the idea are to provide (and *not impose*) to the user a way to open the referenced resource.

HTML

```
<a href="https://www.wikipedia.org/" target="OpenWikipediaWindow">
  Wikipedia, a free encyclopedia (opens in another, possibly already existing,
  tab)
</a>
```

JavaScript

```
let windowObjectReference = null; // global variable
function openRequestedTab(url, windowName) {
  if (windowObjectReference === null || windowObjectReference.closed) {
    windowObjectReference = window.open(url, windowName);
  } else {
    windowObjectReference.focus();
  }
}

const link = document.querySelector("a[target='OpenWikipediaWindow']");
link.addEventListener("click", (event) => {
  openRequestedTab(link.href);
  event.preventDefault();
});
```

The above code solves a few usability problems related to links opening popups. The purpose of the `event.preventDefault()` in the code is to cancel the default action of the link: if the event listener for `click` is executed, then there is no need to execute the default action of the link. But if JavaScript support is disabled or non-existent on the user's browser, then the event listener for `click` is ignored, and the browser loads the referenced resource in the target frame or window that has the name `"WikipediaWindowName"`. If no frame nor window has the name `"WikipediaWindowName"`, then the browser will create a new window and name it `"WikipediaWindowName"`.

> **NOTE:** For more details about the `target` attribute, see `<a>` or `<form>`.

Reuse existing windows and avoid `target="_blank"`

Using `"_blank"` as the target attribute value will create several new and unnamed windows on the user's desktop that cannot be recycled or reused. Try to provide a meaningful name to your `target` attribute and reuse such `target` attribute on your page so that a click on another link may load the referenced resource in an already created and rendered window (therefore speeding up the process for the user) and therefore justifying the reason (and user system resources, time spent) for creating a secondary window in the first place. Using a single `target` attribute value and reusing it in links is much more user resources friendly as it only creates one single secondary window, which is recycled.

Here is an example where a secondary window can be opened and reused for other links:

HTML

```
<p>
  <a href="https://www.wikipedia.org/" target="SingleSecondaryWindowName">
    Wikipedia, a free encyclopedia (opens in another, possibly already existing,
    tab)
  </a>
</p>
<p>
  <a
    href="https://support.mozilla.org/products/firefox"
    target="SingleSecondaryWindowName">
    Firefox FAQ (opens in another, possibly already existing, tab)
  </a>
</p>
```

JavaScript

```
let windowObjectReference = null; // global variable
let previousURL; /* global variable that will store the
                    url currently in the secondary window */
function openRequestedSingleTab(url) {
```

```
  if (windowObjectReference === null || windowObjectReference.closed) {
    windowObjectReference = window.open(url, "SingleSecondaryWindowName");
  } else if (previousURL !== url) {
    windowObjectReference = window.open(url, "SingleSecondaryWindowName");
    /* if the resource to load is different,
       then we load it in the already opened secondary window and then
       we bring such window back on top/in front of its parent window. */
    windowObjectReference.focus();
  } else {
    windowObjectReference.focus();
  }
  previousURL = url;
  /* explanation: we store the current url in order to compare url
     in the event of another call of this function. */
}

const links = document.querySelectorAll(
  "a[target='SingleSecondaryWindowName']",
);
for (const link of links) {
  link.addEventListener("click", (event) => {
    openRequestedSingleTab(link.href);
    event.preventDefault();
  });
}
```

Same-origin policy

If the newly opened browsing context does not share the same origin, the opening script will not be able to interact (reading or writing) with the browsing context's content.

```
// Script from example.com
const otherOriginContext = window.open("https://example.org");
// example.com and example.org are not the same origin

console.log(otherOriginContext.origin);
// DOMException: Permission denied to access property "origin" on cross-origin object
```

```
// Script from example.com
const sameOriginContext = window.open("https://example.com");
// This time, the new browsing context has the same origin

console.log(sameOriginContext.origin);
// https://example.com
```

For more information, refer to the Same-origin policy article.

Accessibility concerns

Avoid resorting to window.open()

It is preferable to avoid resorting to `window.open()`, for several reasons:

- Modern browsers offer a popup-blocking feature.
- Modern browsers offer tab-browsing, and tab-capable browser users prefer opening new tabs to opening new windows in most situations.
- Users may use browser built-in features or extensions to choose whether to open a link in a new window, in the same window, in a new tab, the same tab, or in the background. Forcing the opening to happen in a specific way, using `window.open()`, will confuse them and disregard their habits.
- Popups don't have a menu toolbar, whereas new tabs use the user interface of the browser window; therefore, many users prefer tab-browsing because the interface remains stable.

Never use window.open() inline in HTML

Avoid `<a href="#" onclick="window.open(…);">` or `<a href="javascript:window\.open(…)" …>`.

These bogus `href` values cause unexpected behavior when copying/dragging links, opening links in a new tab/window, bookmarking, or when JavaScript is loading, errors, or is disabled. They also convey incorrect semantics to assistive technologies, like screen readers.

If necessary, use a `<button>` element instead. In general, *you should only use a link for navigation to a real URL*.

Always identify links leading to a secondary window

Identify links that will open new windows in a way that helps navigation for users.

```
<a target="WikipediaWindow" href="https://www.wikipedia.org">
  Wikipedia (opens in new tab)
</a>
```

The purpose is to warn users of context changes to minimize confusion on the user's part: changing the current window or popping up new windows can be very disorienting to users (in the case of a popup, no toolbar provides a "Previous" button to get back to the previous window).

When extreme changes in context are explicitly identified before they occur, then the users can determine if they wish to proceed or so they can be prepared for the change: not only they will not be confused or feel disoriented, but more experienced users can better decide how to open such links (in a new window or not, in the same window, in a new tab or not, in "background" or not).

- WebAIM: Links and Hypertext - Hypertext Links
- MDN / Understanding WCAG, Guideline 3.2
- G200: Opening new windows and tabs from a link only when necessary
- G201: Giving users advanced warning when opening a new window

Specifications

Browser compatibility

See also

- `target` attribute documentation:
  - `<a>`
  - `<form>`
- `window.close()`
- `window.closed`
- `window.focus()`

- `window.opener`
- `rel="opener"` and `rel="noopener"`
- Same-origin policy

□□□ #3
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:35:15 UTC