The `Window` interface's **`open()`** method loads the specified resource into the new or existing browsing context (window, `<iframe>` or tab) with the specified name. If the name doesn't exist, then a new browsing context is opened in a new tab or a new window, and the specified resource is loaded into it.

Syntax

```
var window = window.open(url, windowName, [windowFeatures]);
```

Parameters

`url`
A `DOMString` indicating the URL of the resource to be loaded. This can be a path or URL to an HTML page, image file, or any other resource that is supported by the browser. If the empty string ("") is specified as *`url`*, a blank page is opened into the targeted browsing context.

`windowName` Optional
A `DOMString` specifying the name of the browsing context (window, `<iframe>` or tab) into which to load the specified resource; if the name doesn't indicate an existing context, a new window is created and is given the name specified by *`windowName`*.

This name can be used as the target for the `target` attribute of `<a>` or `<form>` elements. The name should not contain whitespace. Keep in mind that this will NOT be used as the window's title.

`windowFeatures` Optional
A `DOMString` containing a comma-separated list of window features given with their corresponding values in the form "name=value". These features include options such as the window's default size and position, whether or not to include toolbar, and so forth. There must be no whitespace in the string. See Window features below for documentation of each of the features that can be specified.

Return value

A `WindowProxy` object, which is basically a thin wrapper for the `Window` object representing the newly created window, and has all its features available. If the window couldn't be opened, the returned value is instead `null`. The returned reference can be used to access properties and methods of the new window as long as it complies with Same-origin policy security requirements.

Description

The `open()` method creates a new secondary browser window, similar to choosing New Window from the File menu. The *`url`* parameter specifies the URL to be fetched and loaded in the new window. If *`url`* is an empty string, then a new blank, empty window (URL `about:blank`) is created with the default toolbars of the main window.

Note that remote URLs won't load immediately. When `window.open()` returns, the window always contains `about:blank`. The actual fetching of the URL is deferred and starts after the current script block finishes executing. The window creation and the loading of the referenced resource are done asynchronously.

Examples

```
var windowObjectReference;
var windowFeatures = "menubar=yes,location=yes,resizable=yes,scrollbars

function openRequestedPopup() {


  windowObjectReference = window.open("http://www.cnn.com/", "CNN_Windo
}
```

```
var windowObjectReference;

function openRequestedPopup() {
  windowObjectReference = window.open(
    "http://www.domainname.ext/path/ImageFile.png",
    "DescriptiveWindowName",
    "resizable,scrollbars,status"
  );
}
```

If a window with the name already exists, then *`url`* is loaded into the *existing* window. In this case the return value of the method is the existing window and *`windowFeatures`* is ignored. Providing

an empty string for *url* is a way to get a reference to an open window by its name without changing the window's location. On Firefox and Chrome (at least), *this only works from the same parent*, ie. if the current window is the opener of the window you try to get an handle on. Otherwise the call to `window.open()` will just create a new window.

To open a *new* window on every call of `window.open()`, use the special value `_blank` for *windowName*.

> Note on the use of `window.open()` to reopen an existing window with name *windowName* : This functionality is not valid for all browsers and more with variable conditions. ~~Firefox (50.0.1) functions as described: from the same domain+port reopen with same name will access the previously created window~~. Google Chrome (55.0.2883.87 m) on the other hand will do it only from the same parent (as if the window was created dependent, which is the "opener"). This is a wide limitation and generates unbelievable complexity of development. Firefox (51.) gets the handle but cannot run any `Element.focus()` while Chrome can run `focus()` from opener to child but not between siblings nor, reverse, from child to opener. This function is the lonely key to get back the handle on a window if the developer has access only to its name (the name can be saved with cookies or local storage but not the window object handle). For now 10/01/2017 the differencies of behavior found recently have not still been tested for others Browsers.

Window features

windowFeatures is an optional string containing a comma-separated list of requested features of the new window. After a window is opened, JavaScript can't be used to change the features. If *windowName* does not specify an existing window and the *windowFeatures* parameter is not provided (or if the *windowFeatures* parameter is an empty string), then the new secondary window will render the default toolbars of the main window.

> **Tip**: Note that in some browsers, users can override the *windowFeatures* settings and enable (or prevent the disabling of) features

Position and size features

windowFeatures parameter can specify the position and size of the new window.

Note on position and dimension error correction

Position

> If only one of them is specified, the behavior is implementation-dependent, and web author should not rely on it.

`left` or `screenX`
Specifies the distance the new window is placed from the left side of the work area for applications of the user's operating system to the leftmost border (resizing handle) of the browser window. The new window cannot be initially positioned offscreen.

`top` or `screenY`
Specifies the distance the new window is placed from the top side of the work area for applications of the user's operating system to the topmost border (resizing handle) of the browser window. The new window cannot be initially positioned offscreen.

If the *`windowFeatures`* parameter is non-empty and if no position features are defined, then the left and top coordinates of the new window dimension will be 22 pixels from where the most recently rendered window was. An offset is universally implemented by browser manufacturers (it is 29 pixels in IE6 SP2 with the default theme) and its purpose is to help users to notice new windows opening. If the most recently used window was maximized, then there is no offset: the new window will be maximized as well.

Size

> If only one of them is specified, the behavior is implementation-dependent, and web author should not rely on it.

`width` or `innerWidth`
Specifies the width of the content area, viewing area of the new secondary window in pixels. The width value includes the width of the vertical scrollbar if present. The width value does not include the sidebar if it is expanded. The minimum required value is 100.

`height` or `innerHeight`
Specifies the height of the content area, viewing area of the new secondary window in pixels. The height value includes the height of the horizontal scrollbar if present. The height value does not include other UI parts such as location bar, title bar, tab bar, etc. The minimum required value is 100.

If the *`windowFeatures`* parameter is non-empty and no size features are defined, then the new window dimensions will be the same as the dimensions of the most recently rendered window.

Browser-dependent size features

> Do not use them.

`outerWidth` (only on Firefox, obsolete from Firefox 80)
Specifies the width of the whole browser window in pixels. This `outerWidth` value includes the window vertical scrollbar (if present) and left and right window resizing borders.

`outerHeight` (only on Firefox, obsolete from Firefox 80)
Specifies the height of the whole browser window in pixels. This `outerHeight` value includes any/all present toolbar, window horizontal scrollbar (if present) and top and bottom window resizing borders. Minimal required value is 100.

Toolbar and UI parts features

> In modern browsers (Firefox 76 or newer, Google Chrome, Safari, Chromium Edge), the following features are just a condition for whether to open popup or not. See popup condition section.

The following features control the visibility of each UI parts, All features can be set to `yes` or `1`, or just be present to be on. Set them to `no` or `0`, or in most cases just omit them, to be off.

Example: `status=yes`, `status=1`, and `status` have identical results.

`menubar`
If this feature is on, then the new secondary window renders the menubar.

If *`windowFeatures`* is non-empty, `menubar` detaults to off.

`toolbar`
If this feature is on, then the new secondary window renders the toolbar buttons (Back, Forward, Reload, Stop buttons).

In addition to the toolbar buttons, Firefox (before 76) will render the Tab Bar if it is visible, present in the parent window. (If this feature is set to off, all toolbars in the window will be invisible).

If *`windowFeatures`* is non-empty, `toolbar` detaults to off.

`location`
If this feature is on, then the new secondary window renders the location bar or the address bar.

If *`windowFeatures`* is non-empty, `location` detaults to off.

`status`
If this feature is on, then the new secondary window has a status bar.

If *windowFeatures* is non-empty, `status` detaults to off.

`resizable`
If this feature is on, the new secondary window will be resizable.

If *windowFeatures* is non-empty, `resizable` detaults to on.

> **Tip**: For accessibility reasons, it is strongly recommended to set this feature always on

`scrollbars`
If this feature is on, the new secondary window will show horizontal and/or vertical scrollbar(s) if the document doesn't fit into the window's viewport.

If *windowFeatures* is non-empty, `scrollbars` detaults to off.

See note on scrollbars.

> **Tip**: For accessibility reasons, it is strongly encouraged to set this feature always on

Window functionality features

`noopener`
If this feature is set, the newly-opened window will open as normal, except that it will not have access back to the originating window (via `Window.opener` — it returns `null`). In addition, the `window.open()` call will also return `null`, so the originating window will not have access to the new one either. This is useful for preventing untrusted sites opened via `window.open()` from tampering with the originating window, and vice versa.

Note that when `noopener` is used, nonempty target names other than `_top`, `_self`, and `_parent` are all treated like `_blank` in terms of deciding whether to open a new window/tab.

This is supported in modern browsers including Chrome, and Firefox 52+.
See `rel="noopener"` for more information and for browser compatibility details, including information about ancillary effects.

`noreferrer`
If this feature is set, the request to load the content located at the specified URL will be loaded with the request's `referrer` set to `noreferrer`; this prevents the request from sending the URL of the page that initiated the request to the server where the request is sent. In addition, setting this feature also automatically sets `noopener`. See `rel="noreferrer"` for additional details and compatibility information. Firefox introduced support for `noreferrer` in Firefox 68.

Best practices

```
<script type="text/javascript">
var windowObjectReference = null; // global variable

function openFFPromotionPopup() {
  if(windowObjectReference == null || windowObjectReference.closed)
  /* if the pointer to the window object in memory does not exist
     or if such pointer exists but the window was closed */

  {
    windowObjectReference = window.open("http://www.spreadfirefox.com/"
   "PromoteFirefoxWindowName", "resizable,scrollbars,status");
    /* then create it. The new window will be created and
       will be brought on top of any other window. */
  }
  else
  {
    windowObjectReference.focus();
    /* else the window reference must exist and the window
       is not closed; therefore, we can bring it back on top of any oth
       window with the focus() method. There would be no need to re-cre
       the window or to reload the referenced resource. */
  };
}
</script>

(...)

<p><a
 href="http://www.spreadfirefox.com/"
 target="PromoteFirefoxWindowName"
 onclick="openFFPromotionPopup(); return false;"


 title="This link will create a new window or will re-use an already op
>Promote Firefox adoption</a></p>
```

The above code solves a few usability problems related to links opening secondary window. The purpose of the `return false` in the code is to cancel default action of the link: if the onclick event handler 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 onclick event handler is ignored and the browser loads the referenced resource in the target frame or window that has the name "PromoteFirefoxWindowName". If no frame nor window has the name "PromoteFirefoxWindowName", then the browser will create a new window and will name it "PromoteFirefoxWindowName".

More reading on the use of the target attribute:

HTML 4.01 Target attribute specifications

How do I create a link that opens a new window?

You can also parameterize the function to make it versatile, functional in more situations, therefore re-usable in scripts and webpages:

```
<script type="text/javascript">
var windowObjectReference = null; // global variable

function openRequestedPopup(url, windowName) {
  if(windowObjectReference == null || windowObjectReference.closed) {
    windowObjectReference = window.open(url, windowName,
           "resizable,scrollbars,status");
  } else {
    windowObjectReference.focus();

  };
}
</script>

(...)

<p><a
 href="http://www.spreadfirefox.com/"
 target="PromoteFirefoxWindow"
 onclick="openRequestedPopup(this.href, this.target); return false;"
 title="This link will create a new window or will re-use an already op
>Promote Firefox adoption</a></p>
```

You can also make such function able to open only 1 secondary window and to reuse such single secondary window for other links in this manner:

```
<script type="text/javascript">
var windowObjectReference = null; // global variable
var PreviousUrl; /* global variable that will store the
                    url currently in the secondary window */

function openRequestedSinglePopup(url) {
  if(windowObjectReference == null || windowObjectReference.closed) {
    windowObjectReference = window.open(url, "SingleSecondaryWindowName
         "resizable,scrollbars,status");
  } else if(PreviousUrl != url) {
    windowObjectReference = window.open(url, "SingleSecondaryWindowName
      "resizable=yes,scrollbars=yes,status=yes");
    /* 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. */
}
</script>

(...)

<p><a
 href="http://www.spreadfirefox.com/"
 target="SingleSecondaryWindowName"
 onclick="openRequestedSinglePopup(this.href); return false;"
 title="This link will create a new window or will re-use an already op
>Promote Firefox adoption</a></p>


<p><a
 href="http://www.mozilla.org/support/firefox/faq"
```

```
      target="SingleSecondaryWindowName"
      onclick="openRequestedSinglePopup(this.href); return false;"
      title="This link will create a new window or will re-use an already op
     >Firefox FAQ</a></p>
```

FAQ

How can I prevent the confirmation message asking the user whether they want to close the window?

You cannot. **New windows not opened by javascript cannot as a rule be closed by JavaScript.** The JavaScript Console in Mozilla-based browsers will report the warning message: "`Scripts may not close windows that were not opened by script.`" Otherwise the history of URLs visited during the browser session would be lost.

More on the `window.close()` method

How can I bring back the window if it is minimized or behind another window?

First check for the existence of the window object reference of such window and if it exists and if it has not been closed, then use the focus() method. There is no other reliable way. You can examine an example explaining how to use the focus() method.

How do I force a maximized window?

You cannot. All browser manufacturers try to make the opening of new secondary windows noticed by users and noticeable by users to avoid confusion, to avoid disorienting users.

How do I turn off window resizability or remove toolbars?

You cannot force this. Users with Mozilla-based browsers have absolute control over window functionalities like resizability, scrollability and toolbars presence via user preferences in `about:config`. Since your users are the ones who are supposed to use such windows (and not you, being the web author), the best is to avoid interfering with their habits and preferences. We recommend to always set the resizability and scrollbars presence (if needed) to yes to insure accessibility to content and usability of windows. This is in the best interests of both the web author and the users.

How do I resize a window to fit its content?

You cannot reliably because the users can prevent the window from being resized by setting `dom.disable_window_move_resize` to `true` in `about:config` or by editing accordingly their user.js file.

In general, users usually disable moving and resizing of existing windows because allowing authors' scripts to do so has been abused overwhelmingly in the past and the rare scripts that do not abuse such feature are often wrong, inaccurate when resizing the window. 99% of all

those scripts disable window resizability and disable scrollbars when in fact they should enable both of these features to allow a cautious and sane fallback mechanism if their calculations are wrong.

The window method sizeToContent() can also be disabled. Moving and resizing a window remotely on the user's screen via script will very often annoy the users, will disorient the user, and will be wrong at best. The web author expects to have full control of (and can decide about) every position and size aspects of the users' browser window ... which is simply not true.

How do I open a referenced resource of a link in a new tab? or in a specific tab?

To open a resource in a new tab see Tabbed browser. Some Code snippets are available. If you are using the SDK, tabs are handled a bit differently

K-meleon 1.1, a Mozilla-based browser, gives complete control and power to the user regarding how links are opened. Only the user can set his advanced preferences to do that. Some advanced extensions also give Mozilla and Firefox a lot of power over how referenced resources are loaded.

In a few years, the target property of the CSS3 hyperlink module may be implemented (if CSS3 Hyperlink module as it is right now is approved). And even if and when this happens, you can expect developers of browsers with tab-browsing to give the user entire veto power and full control over how links can open web pages. How to open a link should always be entirely under the control of the user.

How do I know whether a window I opened is still open?

You can test for the existence of the window object reference which is the returned value in case of success of the window.open() call and then verify that *windowObjectReference*.closed return value is *false*.

How can I tell when my window was blocked by a popup blocker?

With the built-in popup blockers, you have to check the return value of `window.open()` : it will be `null` if the window wasn't allowed to open. However, for most other popup blockers, there is no reliable way.

What is the JavaScript relationship between the main window and the secondary window?

The `window.open()` method gives a main window a reference to a secondary window; the opener property gives a secondary window a reference to its main window.

I cannot access the properties of the new secondary window. I always get an error in the javascript console saying "Error: uncaught exception: Permission denied to get property <property_name or method_name>. Why is that?

It is because of the cross-domain script security restriction (also referred as the "Same Origin Policy"). A script loaded in a window (or frame) from a distinct origin (domain name) **cannot get nor set** properties of another window (or frame) or the properties of any of its HTML objects coming from another distinct origin (domain name). Therefore, before executing a script targeting a secondary window, the browser in the main window will verify that the secondary window has the same domain name.