**FRAMES AND WINDOWS**

**Popups and Window Methods**

A popup window is one of the oldest methods to show the additional documents to the user.

Basically, you just run:

window.open('https://javascript.info/')

…And it will open a new window with a given URL.

Popups exist from really ancient times, the initial idea was to show another content without closing the main window.

As of now, there are other ways to do that: we can load content dynamically with fetch and show it in a dynamically generated <div>.

Popups are tricky on mobile devices, that don’t show multiple windows simultaneously.

Still, there are tasks where popups are still used, e.g. for OAuth authorization (login with Google/Facebook/…), because:

-A popup is a separate window with its own independent JavaScript environment. So opening a popup with a third-party non-trusted site is safe.

-It’s very easy to open a popup.

-A popup can navigate (change URL) and send messages to the opener window.

**Popup blocking**

Popup blocking is a feature in web browsers and other software applications that prevents unwanted pop-up windows from appearing on a user's screen while they are browsing the internet or using an application.

Pop-up windows are typically small, secondary browser windows that open without the user's explicit request and often contain advertisements or other unsolicited content.

In the past, evil sites abused popups a lot. A bad page could open tons of popup windows with ads. So now most browsers try to block popups and protect the user.

Most browsers block popups if they are called outside of user-triggered event handlers like onclick.

// popup blocked

window.open('https://javascript.info');

// popup allowed

button.onclick = () => {

  window.open('https://javascript.info');

};

This way users are somewhat protected from unwanted popups, but the functionality is not disabled totally.

What if the popup opens from onclick, but after setTimeout? That’s a bit tricky.

**Instructions**

Try this code:

// open after 3 seconds

setTimeout(() => window.open('http://google.com'), 3000);

The popup opens in Chrome, but gets blocked in Firefox.

…If we decrease the delay, the popup works in Firefox too:

// open after 1 seconds

setTimeout(() => window.open('http://google.com'), 1000);

The difference is that Firefox treats a timeout of 2000ms or less are acceptable, but after it – removes the “trust”, assuming that now it’s “outside of the user action”. So the first one is blocked, and the second one is not.

window.open() is a JavaScript method that is used to open a new browser window or tab, depending on the browser's settings and the code used. It allows web developers to create new browser windows or tabs with specified content, dimensions, and other attributes.

The syntax to open a popup is: window.open(url, name, params):

**url**

An URL to load into the new window.

**name**

A name of the new window. Each window has a window.name, and here we can specify which window to use for the popup. If there’s already a window with such name – the given URL opens in it, otherwise a new window is opened.

**params**

The configuration string for the new window. It contains settings, delimited by a comma. There must be no spaces in params, for instance: width:200,height=100.

**Settings for params:**

**Position:**

left/top (numeric) – coordinates of the window top-left corner on the screen. There is a limitation: a new window cannot be positioned offscreen.

width/height (numeric) – width and height of a new window. There is a limit on minimal width/height, so it’s impossible to create an invisible window.

**Window features:**

* + menubar (yes/no) – shows or hides the browser menu on the new window.
  + toolbar (yes/no) – shows or hides the browser navigation bar (back, forward, reload etc) on the new window.
  + location (yes/no) – shows or hides the URL field in the new window. FF and IE don’t allow to hide it by default.
  + status (yes/no) – shows or hides the status bar. Again, most browsers force it to show.
  + resizable (yes/no) – allows to disable the resize for the new window. Not recommended.
  + scrollbars (yes/no) – allows to disable the scrollbars for the new window. Not recommended.
  + There is also a number of less supported browser-specific features, which are usually not used. Check [window.open](https://developer.mozilla.org/en-US/docs/Web/API/Window/open)in MDN for examples.

**A Minimalistic Window**

A minimalistic window refers to a browser window or a pop-up window created using JavaScript with a simplified and minimal set of features and user interface elements. JavaScript can be used to open new browser windows or tabs with customized features and content, and a minimalistic window is one that intentionally keeps the user interface elements and features to a minimum.

1. Let’s open a window with minimal set of features just to see which of them browser allows to disable:

let params = `scrollbars=no,resizable=no,status=no,location=no,toolbar=no,menubar=no,

width=0,height=0,left=-1000,top=-1000`;

open('/', 'test', params);

Here most “window features” are disabled and window is positioned offscreen. Run it and see what really happens. Most browsers “fix” odd things like zero width/height and offscreen left/top. For instance, Chrome open such a window with full width/height, so that it occupies the full screen.

2. Let’s add normal positioning options and reasonable width, height, left, top coordinates:

let params = `scrollbars=no,resizable=no,status=no,location=no,toolbar=no,menubar=no,

width=600,height=300,left=100,top=100`;

open('/', 'test', params);

Rules for omitted settings:

* If there is no 3rd argument in the open call, or it is empty, then the default window parameters are used.
* If there is a string of params, but some yes/no features are omitted, then the omitted features assumed to have no value. So if you specify params, make sure you explicitly set all required features to yes.
* If there is no left/top in params, then the browser tries to open a new window near the last opened window.
* If there is no width/height, then the new window will be the same size as the last opened.

**Accessing Popup from Window**

The open call returns a reference to the new window. It can be used to manipulate it’s properties, change location and even more.

In this example, we generate popup content from JavaScript:

let newWin = window.open("about:blank", "hello", "width=200,height=200");

newWin.document.write("Hello, world!");

And here we modify the contents after loading:

let newWindow = open('/', 'example', 'width=300,height=300')

newWindow.focus();

alert(newWin.location.href); // (\*) about:blank, loading hasn't started yet

newWindow.onload = function() {

  let html = `<div style="font-size:30px">Welcome!</div>`;

  newWindow.document.body.insertAdjacentHTML('afterbegin', html);

};

Please note: immediately after window.open, the new window isn’t loaded yet. That’s demonstrated by alert in line (\*). So we wait for onload to modify it. We could also use DOMContentLoaded handler for newWin.document.

#### Same origin policy

Windows may freely access content of each other only if they come from the same origin (the same protocol://domain:port).

A popup may access the “opener” window as well using window.opener reference. It is null for all windows except popups.

If you run the code below, it replaces the opener (current) window content with “Test”:

let newWin = window.open("about:blank", "hello", "width=200,height=200");

newWin.document.write(

  "<script>window.opener.document.body.innerHTML = 'Test'<\/script>"

);

**Closing a popup**

To close a window: win.close().

To check if a window is closed: win.closed.

Technically, the close() method is available for any window, but window.close() is ignored by most browsers if window is not created with window.open(). So it’ll only work on a popup.

The closed property is true if the window is closed. That’s useful to check if the popup (or the main window) is still open or not. A user can close it anytime, and our code should take that possibility into account.

**Scrolling and resizing**

There are methods to move/resize a window:

**win.moveBy(x,y)**

Move the window relative to current position x pixels to the right and y pixels down. Negative values are allowed (to move left/up).

**win.moveTo(x,y)**

Move the window to coordinates (x,y) on the screen.

**win.resizeBy(width,height)**

Resize the window by given width/height relative to the current size. Negative values are allowed.

**win.resizeTo(width,height)**

Resize the window to the given size.

There’s also window.onresize event.

#### Only popups

To prevent abuse, the browser usually blocks these methods. They only work reliably on popups that we opened, that have no additional tabs.

#### No minification/maximization

JavaScript has no way to minify or maximize a window. These OS-level functions are hidden from Frontend-developers.

Move/resize methods do not work for maximized/minimized windows.

**Scrolling a window**

win.scrollBy(x,y)

Scroll the window x pixels right and y down relative the current scroll. Negative values are allowed.

win.scrollTo(x,y)

Scroll the window to the given coordinates (x,y).

elem.scrollIntoView(top = true)

Scroll the window to make elem show up at the top (the default) or at the bottom for elem.scrollIntoView(false).

There’s also window.onscroll event.

**Focus/Blur on a window**

Theoretically, there are window.focus() and window.blur() methods to focus/unfocus on a window. Also there are focus/blur events that allow to focus a window and catch the moment when the visitor switches elsewhere.

In the past evil pages abused those. For instance, look at this code:

window.onblur = () => window.focus();

When a user attempts to switch out of the window (blur), it brings it back to focus. The intention is to “lock” the user within the window.

So, there are limitations that forbid the code like that. There are many limitations to protect the user from ads and evils pages. They depend on the browser.

For instance, a mobile browser usually ignores that call completely. Also focusing doesn’t work when a popup opens in a separate tab rather than a new window.

Still, there are some things that can be done.

**For instance:**

* When we open a popup, it’s might be a good idea to run a newWindow.focus() on it. Just in case, for some OS/browser combinations it ensures that the user is in the new window now.
* If we want to track when a visitor actually uses our web-app, we can track window.onfocus/onblur. That allows us to suspend/resume in-page activities, animations etc. But please note that the blur event means that the visitor switched out from the window, but they still may observe it. The window is in the background, but still may be visible.