Dialog

- Modal dialogs
- Non-modal dialogs
- Closing a dialog
- Additional details
  - Don't use tabindex
  - ARIA roles
  - CSS defaults

A modal dialog is a special type of pop-up box on a web page: a pop-up that interrupts the user to focus on itself. There are some valid use cases for popping up a dialog, but great consideration should be made before doing so. Modal dialogs force users to focus on specific content, and, temporarily at least, ignore the rest of the page.

Dialogs can be either modal (only the content in the dialog can be interacted with) or non-modal (it's still possible to interact with content outside of the dialog). Modal dialogs are displayed on top of the rest of the page content. The rest of the page is inert and, by default, obscured by a semi-transparent backdrop.

The semantic HTML `<dialog>` element to create a dialog comes with semantics, keyboard interactions, and all the properties and methods of the `HTMLDialogElement` interface.

Modal dialogs

Here is an example of a modal `<dialog>`. Open the dialog with the "Open modal dialog" button. Once opened, there are three ways to close the dialog: the escape key, submitting a form with a button that has the `formmethod="dialog"` set (or if the form itself has `method="dialog"` set), and the `HTMLDialogElement.close()` method.

{% Codepen { user: 'web-dot-dev', id: 'BaOBLNy' } %}

The `HTMLDialogElement` has three main methods, along with all the methods inherited from `HTMLElement`.

```
dialog.show() /* opens the dialog */
dialog.showModal() /* opens the dialog as a modal */
dialog.close() /* closes the dialog */
```

Because this `<dialog>` was opened via the `HTMLDialogElement.showModal()` method, it is a modal dialog. Opening a modal dialog deactivates and obscures everything other than the dialog itself. If you hover over the UI outside of the dialog, you'll note all the elements are behaving as if `pointer-events: none;` was set; even the button that opens the dialog doesn't react to interactions.

When the dialog is opened, focus moves into the dialog. Focus is set on the first element in the sequential keyboard navigation order within that dialog. If you hit the `tab` key repeatedly, you'll note that only the content within the dialog can get focus while the modal dialog is open. Everything outside of the modal dialog is inert as long as the dialog is open.

When a dialog is closed, modal or not, focus is returned to the element that opened the dialog. If you programmatically open a dialog not based on user action, reconsider. If you must, ensure that focus is put back where it was prior to the dialog opening, especially if the user dismisses the dialog without interacting with it.

There is a global `inert` attribute that can be used to disable an element and all of its descendants, other than any active dialog. When a modal dialog is opened using `openModal()`, the inertness or deactivation comes for free; the attribute isn't explicitly set.

The backdrop that obscures everything other than the dialog can be styled using the `::backdrop` pseudo-element. The backdrop is only displayed when a `<dialog>` is displayed with the `.showModal()` method. This pseudo-element matches all the backdrops, including the one displayed when the FullScreen API is used, such as when viewing a video in full-screen mode which doesn't have the same aspect ratio as the screen or monitor.

Non-modal dialogs

The `HTMLDialogElement.show()` similarly opens a dialog, but without adding a backdrop or causing anything to become inert. The escape key does not close non-modal dialogs. Because of this, it is even more important to ensure you include a method of closing the non-modal dialog. In doing so, if the closer is outside the dialog, realize the focus will go to the element that opened the dialog, which may not be the best user experience.

{% Codepen { user: 'web-dot-dev', id: 'bGKQvza' } %}

While a button to close the dialog is not officially required by the specification, consider it as required. The escape key will close a modal dialog, but not a non-modal one. A visible button that is able to receive focus improves accessibility and user experience.

Closing a dialog

You don’t need the `HTMLDialogElement.close()` method to close a dialog. You don’t need JavaScript at all. To close the `<dialog>` without JavaScript, include a form with a dialog method by either setting `method="dialog"` on the `<form>` or `formmethod="dialog"` on the button.

When a user submits via the `dialog` method, the state of user-entered data is maintained. While there is a submit event—the form goes through constraint validation (unless `novalidate` is set)—the user data is neither cleared nor submitted. A close button without JavaScript can be written as:

```
<dialog open>
  <form method="dialog">
    <button type="submit" autofocus>close</button>
  </form>
</dialog>
```

You may have noticed the `autofocus` attribute set on the close `<button>` in this example. Elements with `autofocus` attribute set within a `<dialog>` will not receive focus on page load (unless the page is loaded with the dialog visible). They will, however, get focus when the dialog is opened.

By default, when a dialog is opened, the first focusable element within the dialog will receive focus unless a different element within the dialog has the `autofocus` attribute set. Setting the `autofocus` attribute on the close button ensures it receives focus when the dialog is opened. But including `autofocus` within a `<dialog>` should only be done with much consideration. All the elements in the sequence coming before the autofocused element are skipped. We discuss this attribute further in the focus lesson.

The `HTMLDialogElement` interface includes a `returnValue` property. Submitting a form with a `method="dialog"` sets the `returnValue` to the `name`, if any, of the submit button used to submit the form. If we had written `<button type="submit" name="toasty">close</button>`, the `returnValue` would be `toasty`.

When a dialog is opened, the boolean `open` attribute is present, meaning the dialog is active and can be interacted with. When a dialog is opened by adding the `open` attribute rather than via `.show()` or `.showModal()`, the dialog will be modal-less. The `HTMLDialogElement.open` property returns `true` or `false`, depending on whether the dialog is available for interaction—not whether it is modal or not.

While JavaScript is the preferred method of opening a dialog, including the `open` attribute on page load, and then removing it with `.close()`, can help ensure the dialog is available even when JavaScript is not.

Additional details

Don't use `tabindex`

The element that is activated to open the dialog and the close button contained in it (and possibly other content) can receive focus and are interactive. The `<dialog>` element is not interactive and doesn’t receive focus. Do not add the `tabindex` property to the dialog itself.

ARIA roles

The implicit role is `dialog`. If the dialog is a confirmation window communicating an important message that requires a confirmation or other user response, set `role="alertdialog"`. The dialog should also have an accessible name. If visible text can provide for the accessible name, add `aria-labelledby="idOfLabelingText"`.

CSS defaults

Note that browsers provide default styling for `dialog`. Firefox, Chrome, and Edge set `color: CanvasText;` `background-color: Canvas;` and Safari sets `color: black; background-color: white;` in their user-agent stylesheets. The `color` is inherited from `dialog` and not from `body` or `:root`, which may be unexpected. The `background-color` property is not inherited.

{% Assessment 'dialog' %}

□□□ #2
□ □□□ □□□ 2026-10-31 22:51:03 UTC
□ □□□ □□□ 2026-10-31 22:51:04 UTC