Details and summary

- Toggling visibility: the open attribute
- Toggling the summary marker
- How errors are handled
- The HTMLDetailsElement interface

A disclosure widget is a user interface control that hides and shows content. If you are reading this on web.dev, and your viewport is less than 106 ems wide, clicking on the "On this page" above this paragraph reveals the table of contents for this section. If you don't see it, shrink the browser to view the table of contents navigation on this page as a disclosure widget.

The accordion graphical user interface is a series of vertically stacked disclosure widgets. A common use case for the accordion UI is the Frequently Asked Questions (FAQ) page on many sites. An accordion FAQ contains a list of visible questions; clicking on a question expands, or "discloses", the answer to that question.

jQuery has included an accordion UI pattern since at least 2009. The original JavaScript-free accordion solution included making each FAQ question a `<label>` followed by the checkmark it labeled, and then displaying `<div>` answer when the checkmark was checked. The CSS looked something like this:

```
#FAQ [type="checkbox"] + div.answer {
  /* all the answer styles */
  display: none;
}
#FAQ [type="checkbox"]:checked + div.answer {
  display: block;
}
```

Why the history? Disclosure widgets, such as accordions, without JavaScript or form control hacks, are a relatively recent addition; the `<details>` and `<summary>` elements have only been fully supported in all modern browsers since January 2020. You can now create functioning, albeit less than attractive, disclosure widgets using only semantic HTML. The `<details>` and `<summary>` elements are all you need: they are a built-in way to handle expanding and collapsing content. When a user clicks or taps a `<summary>`, or releases the `Enter` key when the `<summary>` has focus,

the contents of the parent `<details>` toggle to visible!

https://codepen.io/web-dot-dev/pen/MWqgjQr

Like all semantic content, you can progressively enhance the default features and appearance. In this case, a tiny bit of CSS has been added, but nothing else:

https://codepen.io/web-dot-dev/pen/ExeYgQd

You'll note, these Codepens contain no JavaScript.

Toggling visibility: the `open` attribute

The `<details>` element is the disclosure widget container. The `<summary>` is the summary or legend for its parent `<details>`. The summary is always displayed, acting as a button that toggles the display of the rest of the parent’s contents. Interacting with the `<summary>` toggles the display of the self-labeled summary siblings by toggling the `<details>`' element's `open` attribute.

The `open` attribute is a boolean attribute. If present, no matter the value or lack thereof, it indicates that all the `<details>` contents are shown to the user. If the `open` attribute is not present, only the contents of the `<summary>` are shown.

Because the `open` attribute is added and removed automatically as the user interacts with the control, it can be used in CSS to style the element differently based on its state.

You can create an accordion with a list of multiple `<details>` elements, each with a `<summary>` child. Omitting the `open` attribute in your HTML means the `<details>` will all be collapsed, or closed, with just the summary headings visible when the page loads; each heading being the opener for the rest of the contents in the parent `<details>`. If you include the `open` attribute in your HTML, the `<details>` will render expanded, with the contents visible, when the page loads.

The hidden content in the collapsed state is searchable in some browsers but not others, even though the collapsed content is not part of the DOM. If you search in Edge or Chrome, the details containing a search term will expand to display the occurrence. This behavior is not replicated in Firefox or Safari.

The `<summary>` must be the first child of a `<details>` element, representing a summary, caption, or legend for the rest of the contents of the parent `<details>` element in which it is nested. The `<summary>` element's contents can be any heading content, plain text, or HTML that can be used within a paragraph.

Toggling the summary marker

In the two earlier Codepens, you'll note the arrow to the inline-start side of the summary. A disclosure widget is typically presented on-screen using a small triangle that rotates (or twists) to indicate open/closed status, with a label next to the triangle. The contents of the `<summary>` element label the disclosure widget. The rotating arrow at the top of each section is a `::marker` set on the `<summary>` element. Like list items, the `<summary>` element supports the `list-style` shorthand property and its longhand properties, including `list-style-type`. You can style the disclosure triangle with CSS, including changing the marker used from a triangle to any other bullet type, including an image with `list-style-image`.

To apply other styles, use a selector similar to `details summary::marker`. The `::marker` pseudo-element only accepts a limited number of styles. Removing the `::marker` and replacing it with the easier-to-style `::before` is common practice, with CSS styles changing the style of the generated content slightly based on the presence (or absence) of the open attribute. You can remove the disclosure widget icon by setting `list-style: none` or set the content of the marker to `none`, but always include visual indicators to inform sighted users that the summary content is a toggle button that will show and hide content upon activation.

```
details summary::before {
  /* all the styles */
}
details[open] summary::before {
  /* changes applied when open only */
}
```

https://codepen.io/web-dot-dev/pen/jOvNMxL

This example removes the default marker, and adds generated content to create a `+` when the details are closed and a `-` when the details are open.

If you want the details block open by default, include the `open` attribute on the opening `<details>` tag. You can also add space between each dialog and transition the rotation of the marker created with generated content to improve the appearance:

https://codepen.io/web-dot-dev/pen/yLxBajp

How errors are handled

If you don't include a `<summary>`, the browser will create one for you: with a marker and the word "details". This summary is part of a shadow root, and therefore will not have author CSS summary styles applied. Unfortunately, Safari does not include the details in the keyboard focus order.

If you do include a `<summary>`, but it is not the first element in the `<details>`, the browser still displays the summary as it should. It will also not fail if you include a link, label, or other interactive element within the summary, but browsers handle interactive content within interactive content differently. For example, if you include a link in a summary, some browsers will add both the summary and the link to the default tabbing order, but other browsers will not focus on the link by default. If you click on a `<label>` nested in a `<summary>`, some browsers will give focus to the associated form control; other browsers will give focus to the form control and toggle the `<details>` open or closed.

https://codepen.io/web-dot-dev/pen/QWVLKxg

The `HTMLDetailsElement` interface

Like all HTML elements, the `HTMLDetailsElement` inherits all properties, methods, and events from `HTMLElement`, and adds the `open` instance property and a `toggle` event. The `HTMLDetailsElement.open` property is a boolean value reflecting the `open` HTML attribute, indicating whether or not the element's contents (not counting the `<summary>`) are to be shown to the user. The toggle event is fired when the `<details>` element is toggled open or closed. You can listen to this event using `addEventListener()`.

If you want to write a script to close the opened details when the user opens any other details, remove the open attribute using `removeAttribute("open")`:

https://codepen.io/web-dot-dev/pen/PodYGBz

This is the only example to use JavaScript. You probably don't need JavaScript except for this functionality of closing other opened disclosure widgets.

Remember, `<details>` and `<summary>` can be heavily styled and can even be used to create tool tips. But, if you're going to use these semantic elements for use cases in which the native semantics are a mismatch, always ensure that you maintain accessibility. HTML for the most part is by default accessible. Our job as developers is to ensure our content stays accessible.

□□□ #3
□ □□□ □□□ 2026-10-31 22:51:00 UTC
□ cocobook-sync □□□ 2026-11-04 17:39:21 UTC