Overview

Things to know when using the popover plugin:

- Popovers rely on the 3rd party library Popper for positioning. You must include [popper.min.js]({{< param "cdn.popper" >}}) before bootstrap.js or use `bootstrap.bundle.min.js` / `bootstrap.bundle.js` which contains Popper in order for popovers to work!
- Popovers require the [tooltip plugin]({{< docsref "/components/tooltips" >}}) as a dependency.
- Popovers are opt-in for performance reasons, so **you must initialize them yourself**.
- Zero-length `title` and `content` values will never show a popover.
- Specify `container: 'body'` to avoid rendering problems in more complex components (like our input groups, button groups, etc).
- Triggering popovers on hidden elements will not work.
- Popovers for `.disabled` or `disabled` elements must be triggered on a wrapper element.
- When triggered from anchors that wrap across multiple lines, popovers will be centered between the anchors' overall width. Use `.text-nowrap` on your `<a>` s to avoid this behavior.
- Popovers must be hidden before their corresponding elements have been removed from the DOM.
- Popovers can be triggered thanks to an element inside a shadow DOM.

{{< callout info >}} {{< partial "callout-info-sanitizer.md" >}} {{< /callout >}}

{{< callout info >}} {{< partial "callout-info-prefersreducedmotion.md" >}} {{< /callout >}}

Keep reading to see how popovers work with some examples.

Example: Enable popovers everywhere

One way to initialize all popovers on a page would be to select them by their `data-bs-toggle` attribute:

```
var popoverTriggerList = [].slice.call(document.querySelectorAll('[data-bs-toggle="popover"]'))
var popoverList = popoverTriggerList.map(function (popoverTriggerEl) {
  return new bootstrap.Popover(popoverTriggerEl)
})
```

Example: Using the `container` option

When you have some styles on a parent element that interfere with a popover, you'll want to specify a custom `container` so that the popover's HTML appears within that element instead.

```
var popover = new bootstrap.Popover(document.querySelector('.example-popover'), {
  container: 'body'
})
```

Example

{{< example >}} Click to toggle popover {{< /example >}}

Four directions

Four options are available: top, right, bottom, and left aligned. Directions are mirrored when using Bootstrap in RTL.

{{< example >}} Popover on top Popover on right Popover on bottom Popover on left {{< /example >}}

Dismiss on next click

Use the `focus` trigger to dismiss popovers on the user's next click of a different element than the toggle element.

{{< callout danger >}}

Specific markup required for dismiss-on-next-click

For proper cross-browser and cross-platform behavior, you must use the `<a>` tag, *not* the `<button>` tag, and you also must include a `tabindex` attribute. {{< /callout >}}

{{< example >}} Dismissible popover {{< /example >}}

```
var popover = new bootstrap.Popover(document.querySelector('.popover-dismiss'), {
  trigger: 'focus'
})
```

Disabled elements

Elements with the `disabled` attribute aren't interactive, meaning users cannot hover or click them to trigger a popover (or tooltip). As a workaround, you'll want to trigger the popover from a wrapper `<div>` or `<span>`, ideally made keyboard-focusable using `tabindex="0"`.

For disabled popover triggers, you may also prefer `data-bs-trigger="hover focus"` so that the popover appears as immediate visual feedback to your users as they may not expect to *click* on a disabled element.

{{< example >}} Disabled button {{< /example >}}

Sass

Variables

{{< scss-docs name="popover-variables" file="scss/_variables.scss" >}}

Usage

Enable popovers via JavaScript:

```
var exampleEl = document.getElementById('example')
var popover = new bootstrap.Popover(exampleEl, options)
```

{{< callout warning >}}

Making popovers work for keyboard and assistive technology users

To allow keyboard users to activate your popovers, you should only add them to HTML elements that are traditionally keyboard-focusable and interactive (such as links or form controls). Although arbitrary HTML elements (such as `<span>`s) can be made focusable by adding the `tabindex="0"` attribute, this will add potentially annoying and confusing tab stops on non-interactive elements for keyboard users, and most assistive technologies currently do not announce the popover's content in this situation. Additionally, do not rely solely on `hover` as the trigger for your popovers, as this will make them impossible to trigger for keyboard users.

While you can insert rich, structured HTML in popovers with the `html` option, we strongly recommend that you avoid adding an excessive amount of content. The way popovers currently work is that, once displayed, their content is tied to the trigger element with the `aria-describedby` attribute. As a result, the entirety of the popover's content will be announced to assistive technology users as one long, uninterrupted stream.

Additionally, while it is possible to also include interactive controls (such as form elements or links) in your popover (by adding these elements to the `allowList` of allowed attributes and tags), be aware that currently the popover does not manage keyboard focus order. When a keyboard user opens a popover, focus remains on the triggering element, and as the popover usually does not immediately follow the trigger in the document's structure, there is no guarantee that moving forward/pressing `TAB` will move a keyboard user into the popover itself. In short, simply adding interactive controls to a popover is likely to make these controls unreachable/unusable for keyboard users and users of assistive technologies, or at the very least make for an illogical overall focus order. In these cases, consider using a modal dialog instead. {{< /callout >}}

Options

Options can be passed via data attributes or JavaScript. For data attributes, append the option name to `data-bs-`, as in `data-bs-animation=""`. Make sure to change the case type of the option name from camelCase to kebab-case when passing the options via data attributes. For example, instead of using `data-bs-customClass="beautifier"`, use `data-bs-custom-class="beautifier"`.

{{< callout warning >}} Note that for security reasons the `sanitize`, `sanitizeFn`, and `allowList` options cannot be supplied using data attributes. {{< /callout >}}

Name	Type	Default	Description
`animation`	boolean	`true`	Apply a CSS fade transition to the popover
`container`	string | element | false	`false`	Appends the popover to a specific element. Example: `container: 'body'`. This option is particularly useful in that it allows you to position the popover in the flow of the document near the triggering element - which will prevent the popover from floating away from the triggering element during a window resize.
`content`	string | element | function	`''`	Default content value if `data-bs-content` attribute isn't present. If a function is given, it will be called with its `this` reference set to the element that the popover is attached to.
`delay`	number | object	`0`	Delay showing and hiding the popover (ms) - does not apply to manual trigger type

			If a number is supplied, delay is applied to both hide/show Object structure is: `delay: { "show": 500, "hide": 100 }`
`html`	boolean	`false`	Insert HTML into the popover. If false, `innerText` property will be used to insert content into the DOM. Use text if you're worried about XSS attacks.
`placement`	string | function	`'right'`	How to position the popover - auto | top | bottom | left | right. When `auto` is specified, it will dynamically reorient the popover. When a function is used to determine the placement, it is called with the popover DOM node as its first argument and the triggering element DOM node as its second. The `this` context is set to the popover instance.
`selector`	string | false	`false`	If a selector is provided, popover objects will be delegated to the specified targets. In practice, this is used to enable dynamic HTML content to have popovers added. See this and an informative example.
`template`	string	`'<div class="popover" role="tooltip"><div class="popover-arrow"></div><h3 class="popover-header"></h3><div class="popover-body"></div></div>'`	Base HTML to use when creating the popover. The popover's `title` will be injected into the `.popover-header`. The popover's `content` will be injected into the `.popover-body`. `.popover-arrow` will become the popover's arrow. The outermost wrapper element should have the `.popover` class.
`title`	string | element | function	`''`	Default title value if `title` attribute isn't present. If a function is given, it will be called with its `this` reference set to the element that the popover is attached to.

`trigger`	string	`'click'`	How popover is triggered - click | hover | focus | manual. You may pass multiple triggers; separate them with a space. `manual` cannot be combined with any other trigger.
`fallbackPlacements`	array	`['top', 'right', 'bottom', 'left']`	Define fallback placements by providing a list of placements in array (in order of preference). For more information refer to Popper's [behavior docs]()
`boundary`	string | element	`'clippingParents'`	Overflow constraint boundary of the popover (applies only to Popper's preventOverflow modifier). By default it's `'clippingParents'` and can accept an HTMLElement reference (via JavaScript only). For more information refer to Popper's [detectOverflow docs]().
`customClass`	string | function	`''`	Add classes to the popover when it is shown. Note that these classes will be added in addition to any classes specified in the template. To add multiple classes, separate them with spaces: `'class-1 class-2'`. You can also pass a function that should return a single string containing additional class names.
`sanitize`	boolean	`true`	Enable or disable the sanitization. If activated `'template'`, `'content'` and `'title'` options will be sanitized. See the [}}">sanitizer section in our JavaScript documentation]().
`allowList`	object	[}}">Default value]()	Object which contains allowed attributes and tags
`sanitizeFn`	null | function	`null`	Here you can supply your own sanitize function. This can be useful if you prefer to use a dedicated library to perform sanitization.
`offset`	array | string | function	`[0, 8]`	Offset of the popover relative to its target. You can pass a string in data attributes with comma separated values like: `data-bs-offset="10,20"` When a function is used to determine the offset, it is called with an object containing the popper placement, the reference, and

			popper rects as its first argument. The triggering element DOM node is passed as the second argument. The function must return an array with two numbers: `[skidding, distance]`. For more information refer to Popper's offset docs.
`popperConfig`	null | object | function	`null`	To change Bootstrap's default Popper config, see Popper's configuration. When a function is used to create the Popper configuration, it's called with an object that contains the Bootstrap's default Popper configuration. It helps you use and merge the default with your own configuration. The function must return a configuration object for Popper.

{{< callout info >}}

Data attributes for individual popovers

Options for individual popovers can alternatively be specified through the use of data attributes, as explained above. {{< /callout >}}

Using function with `popperConfig`

```
var popover = new bootstrap.Popover(element, {
  popperConfig: function (defaultBsPopperConfig) {
    // var newPopperConfig = {...}
    // use defaultBsPopperConfig if needed...
    // return newPopperConfig
  }
})
```

Methods

{{< callout danger >}} {{< partial "callout-danger-async-methods.md" >}} {{< /callout >}}

show

Reveals an element's popover. **Returns to the caller before the popover has actually been shown** (i.e. before the `shown.bs.popover` event occurs). This is considered a "manual" triggering of the popover. Popovers whose title and content are both zero-length are never displayed.

```
myPopover.show()
```

hide

Hides an element's popover. **Returns to the caller before the popover has actually been hidden** (i.e. before the `hidden.bs.popover` event occurs). This is considered a "manual" triggering of the popover.

```
myPopover.hide()
```

toggle

Toggles an element's popover. **Returns to the caller before the popover has actually been shown or hidden** (i.e. before the `shown.bs.popover` or `hidden.bs.popover` event occurs). This is considered a "manual" triggering of the popover.

```
myPopover.toggle()
```

dispose

Hides and destroys an element's popover (Removes stored data on the DOM element). Popovers that use delegation (which are created using the `selector` option) cannot be individually destroyed on descendant trigger elements.

```
myPopover.dispose()
```

enable

Gives an element's popover the ability to be shown. **Popovers are enabled by default.**

```
myPopover.enable()
```

disable

Removes the ability for an element's popover to be shown. The popover will only be able to be shown if it is re-enabled.

```
myPopover.disable()
```

toggleEnabled

Toggles the ability for an element's popover to be shown or hidden.

```
myPopover.toggleEnabled()
```

update

Updates the position of an element's popover.

```
myPopover.update()
```

getInstance

Static method which allows you to get the popover instance associated with a DOM element

```
var exampleTriggerEl = document.getElementById('example')
var popover = bootstrap.Popover.getInstance(exampleTriggerEl) // Returns a Bootstrap
popover instance
```

getOrCreateInstance

Static method which allows you to get the popover instance associated with a DOM element, or create a new one in case it wasn't initialized

```
var exampleTriggerEl = document.getElementById('example')
var popover = bootstrap.Popover.getOrCreateInstance(exampleTriggerEl) // Returns a
Bootstrap popover instance
```

Events

Event type	Description
show.bs.popover	This event fires immediately when the `show` instance method is called.
shown.bs.popover	This event is fired when the popover has been made visible to the user (will wait for CSS transitions to complete).
hide.bs.popover	This event is fired immediately when the `hide` instance method has been called.
hidden.bs.popover	This event is fired when the popover has finished being hidden from the user (will wait for CSS transitions to complete).
inserted.bs.popover	This event is fired after the `show.bs.popover` event when the popover template has been added to the DOM.

```
var myPopoverTrigger = document.getElementById('myPopover')
myPopoverTrigger.addEventListener('hidden.bs.popover', function () {
  // do something...
})
```