markdown-it-anchor npm v9.2.0

A markdown-it plugin that adds an `id` attribute to headings and optionally permalinks.

English | 中文 (v7.0.1)

Overview

This plugin adds an `id` attribute to headings, e.g. `## Foo` becomes `<h2 id="foo">Foo</h2>`.

Optionally it can also include permalinks, e.g. `<h2 id="foo"><a class="header-anchor" href="#foo">Foo</a></h2>` and a bunch of other variants!

Usage

```
const md = require('markdown-it')()
  .use(require('markdown-it-anchor'),
       opts)
```

See a demo as JSFiddle.

The `opts` object can contain:

Name	Description	Default
`level`	Minimum level to apply anchors, or array of selected levels.	1
`permalink`	A function to render permalinks, see permalinks below.	`undefined`
`slugify`	A custom slugification function.	See `index.js`
`callback`	Called with token and info after rendering.	`undefined`

```
# Watch file changes to update `dist/`.
npm run dev

# Run tests, will use the build version
        so make sure to build after
# making changes.
npm test
```


Name	Description	Default
`getTokensText`	A custom function to get the text contents of the title from its tokens.	See `index.js`
`tabIndex`	Value of the `tabindex` attribute on headings, set to `false` to disable.	`-1`
`uniqueSlugStartIndex`	Index to start with when making duplicate slugs unique.	1

All headers greater than the minimum `level` will have an `id` attribute with a slug of their content. For example, you can set `level` to 2 to add anchors to all headers but `h1`. You can also pass an array of header levels to apply the anchor, like `[2, 3]` to have an anchor on only level 2 and 3 headers.

If a `permalink` renderer is given, it will be called for each matching header to add a permalink. See permalinks below.

If a `slugify` function is given, you can decide how to transform a heading text to a URL slug. See user-friendly URLs.

The `callback` option is a function that will be called at the end of rendering with the `token` and an `info` object. The `info` object has `title` and `slug` properties with the token content and the slug used for the identifier.

We set by default `tabindex="-1"` on headers. This marks the headers as focusable elements that are not reachable by keyboard navigation. The effect is that screen readers will read the title content when it's being jumped to. Outside of screen readers, the experience is the same as not setting that attribute. You can override this behavior with the `tabIndex` option. Set it to `false` to remove the attribute altogether, otherwise the value will be used as attribute value.

Finally, you can customize how the title text is extracted from the markdown-it tokens (to later generate the slug). See user-friendly URLs.

User-friendly URLs

Starting from v5.0.0, markdown-it-anchor dropped the `string` package to retain our core value of being an impartial and secure library. Nevertheless, users looking for backward compatibility may want the old `slugify` function:

```
npm install string
```

```
const string = require('string')
const slugify = s =>
        string(s).slugify().toString()

const md = require('markdown-it')()
  .use(require('markdown-it-anchor'),
        { slugify })
```


This function can freely modify the token stream (`state.tokens`), usually around the given `idx`, to construct the anchor.

Because of the way the token stream works, a `heading_open` token is usually followed by a `inline` token that contains the actual text (and inline markup) of the heading, and finally a `heading_close` token. This is why you'll see most built-in permalink renderers touch `state.tokens[idx + 1]`, because they update the contents of the `inline` token that follows a `heading_open`.

Debugging

If you want to debug this library more easily, we support source maps.

Use the source-map-support module to enable it with Node.js.

```
node -r source-map-support/register
        your-script.js
```

Development

```
# Build the library in the `dist/`
        directory.
npm run build
```

```
  })
})
```

```
<h2 id="title"><a class="header-anchor"
        href="#title" aria-
        hidden="true">#</a> Title</h2>
```

While no experience might be arguably better than a bad experience, I would instead recommend using one of the above renderers to provide an accessible experience. My favorite one is the header link, which is also the simplest one.

Custom permalink

If none of those options suit you, you can always make your own renderer! Take inspiration from the code behind all permalinks.

The signature of the function you pass in the `permalink` option is the following:

```
function renderPermalink (slug, opts,
        state, idx) {}
```

Where `opts` are the markdown-it-anchor options, `state` is a markdown-it `StateCore` instance, and `idx` is the index of the `heading_open` token in the `state.tokens` array. That array contains `Token` objects.

To make sense of the "token stream" and the way token objects are organized, you will probably want to read the markdown-it design principles page.

Another popular library for this is `@sindresorhus/slugify`, which have better Unicode support and other cool features:

```
npm install @sindresorhus/slugify
```

```
const slugify = require('@sindresorhus/
        slugify')

const md = require('markdown-it')()
  .use(require('markdown-it-anchor'), {
        slugify: s => slugify(s) })
```

Custimizing the slugify input

Additionally, if you want to further customize the title that gets passed to the `slugify` function, you can do so by customizing the `getTokensText` function, that gets the plain text from a list of markdown-it inline tokens:

```
function getTokensText (tokens) {
  return tokens
    .filter(token => !['html_inline',
        'image'].includes(token.type))
    .map(t => t.content)
    .join('')
}

const md = require('markdown-it')()
  .use(require('markdown-it-anchor'),
        { getTokensText })
```

By default we include only `text` and `code_inline` tokens, which appeared to be a sensible approach for the vast majority of use cases.

An alternative approach is to include every token's content except for `html_inline` and `image` tokens, which yields the exact same results as the previous approach with a stock markdown-it, but would also include custom tokens added by any of your markdown-it plugins, which might or might not be desirable for you. Now you have the option!

Slugifying with state

If you need access to the markdown-it state from the slugify function, e.g. to access `state.env`, you can use `slugifyWithState` instead.

```
const md = require('markdown-it')()
  .use(require('markdown-it-anchor'), {
    slugifyWithState: (title, state) =>
        `${state.env.id}-$
        {slugify(title)}`
  })
```



```
  Title
</h2>
```

While this example allows more accessible anchors with the same markup as previous versions of markdown-it-anchor, it's still not ideal. The assistive text for permalinks will be read as part of the heading when listing all the titles of the page, e.g. "jump to heading title 1, jump to heading title 2" and so on. Also that assistive text is not very useful when listing the links in the page (which will read "jump to heading, jump to heading, jump to heading" for each of your permalinks).

ARIA hidden

This is just an alias for `linkInsideHeader` with `ariaHidden: true` by default, to mimic GitHub's way of rendering permalinks.

Setting `aria-hidden="true"` makes the permalink explicitly inaccessible instead of having the permalink and its symbol being read by screen readers as part of every single headings (which was a pretty terrible experience).

```
const anchor = require('markdown-it-
        anchor')
const md = require('markdown-it')()

md.use(anchor, {
  permalink:
        anchor.permalink.ariaHidden({
    placement: 'before'
```

Name	Description	Default
`placement`	Placement of the permalink, can be `before` or `after` the header. This option used to be called `permalinkBefore`.	`after`
`ariaHidden`	Whether to add `aria-hidden="true"`, see ARIA hidden.	`false`
	See common options.	

```
const anchor = require('markdown-it-
        anchor')
const md = require('markdown-it')()

md.use(anchor, {
  permalink:
        anchor.permalink.linkInsideHeader({
    symbol: `
      <span class="visually-hidden">Jump
        to heading</span>
      <span aria-hidden="true">#</span>
    `,
    placement: 'before'
  })
})

<h2 id="title">
  <a class="header-anchor"
        href="#title">
    <span class="visually-hidden">Jump
        to heading</span>
    <span aria-hidden="true">#</span>
  </a>
```


Manually setting the `id` attribute

You might want to explicitly set the `id` attribute of your headings from the Markdown document, for example to keep them consistent across translations.

markdown-it-anchor is designed to reuse any existing `id`, making markdown-it-attrs a perfect fit for this use case. Make sure to load it before markdown-it-anchor!

Then you can do something like this:

```
# Your title {#your-custom-id}
```

The anchor link will reuse the `id` that you explicitly defined.

Compatible table of contents plugin

Looking for an automatic table of contents (TOC) generator? Take a look at markdown-it-toc-done-right it's made from the ground to be a great companion of this plugin.

Parsing headings from HTML blocks

markdown-it-anchor doesn't parse HTML blocks, so headings defined in HTML blocks will be ignored. If you need to add anchors to both HTML headings and Markdown headings, the

easiest way would be to do it on the final HTML rather than during the Markdown parsing phase:

```
const { parse } = require('node-html-
        parser')

const root = parse(html)

for (const h of
        root.querySelectorAll('h1, h2,
        h3, h4, h5, h6')) {
  const slug = h.getAttribute('id') ||
        slugify(h.textContent)
  h.setAttribute('id', slug)
  h.innerHTML = `<a href="#${slug}">$
        {h.innerHTML}</a>`
}

console.log(root.toString())
```

Or with a (not accessible) GitHub-style anchor, replace the `h.innerHTML` part with:

```
h.insertAdjacentHTML('afterbegin', `<a
        class="anchor" aria-
        hidden="true" href="#$
        {slug}">🔗</a> `)
```

While this still needs extra work like handling duplicated slugs and IDs, this should give you a solid base.

That said if you really want to use markdown-it-anchor for this even though it's not designed to, you can do like npm does with their marky-markdown parser, and transform the `html_block`


```
<h2 id="title">Title</h2>
<a class="header-anchor" href="#title"
        aria-describedby="title">#</a>
```

Link inside header

This is the equivalent of the default permalink in previous versions. The reason it's not the first one in the list is because this method has accessibility issues.

If you use a symbol like just # without adding any markup around, screen readers will read it as part of every heading (in the case of #, it could be read "pound", "number" or "number sign") meaning that if you title is "my beautiful title", it will read "number sign my beautiful title" for example. For other common symbols, 🔗 is usually read as "link symbol" and ¶ as "pilcrow".

Additionally, screen readers users commonly request the list of all links in the page, so they'll be flooded with "number sign, number sign, number sign" for each of your headings.

I would highly recommend using one of the markups above which have a better experience, but if you really want to use this markup, make sure to pass accessible HTML as `symbol` to make things usable, like in the example below, but even that has some flaws.

With that said, this permalink allows the following options:

Name	Description	Default
`space`	Add a space between the header text and the permalink symbol. Set it to a string to customize the space (e.g. ` `).	`true`

```
md.use(anchor, {
  permalink:
        anchor.permalink.linkAfterHeader({
    style: 'aria-label'
    assistiveText: title => `Permalink
        to “${title}”`
  })
})
```

```
<h2 id="title">Title</h2>
<a class="header-anchor" href="#title"
        aria-label="Permalink to
        “Title”">#</a>
```

`aria-describedby` and `aria-labelledby` variants

This removes the need to customize the assistive text to your locale and doesn't need a visually hidden `span` either, but since the anchor will be described by just the text of the title without any context, it might be confusing.

```
const anchor = require('markdown-it-
        anchor')
const md = require('markdown-it')()

md.use(anchor, {
  permalink:
        anchor.permalink.linkAfterHeader({
    style: 'aria-describedby' // Or
        `aria-labelledby`
  })
})
```


tokens into a sequence of `heading_open`, `inline`, and `heading_close` tokens that can be handled by markdown-it-anchor:

```
const md = require('markdown-it')()
  .use(require('@npmcorp/marky-markdown/
        lib/plugin/html-heading'))
  .use(require('markdown-it-anchor'),
        opts)
```

While they use regexes to parse the HTML and it won't gracefully handle any arbitrary HTML, it should work okay for the happy path, which might be good enough for you.

You might also want to check this implementation which uses Cheerio for a more solid parsing, including support for HTML attributes.

The only edge cases I see it failing with are multiple headings defined in the same HTML block with arbitrary content between them, or headings where the opening and closing tag are defined in separate `html_block` tokens, both which should very rarely happen.

If you need a bulletproof implementation, I would recommend the first HTML parser approach I documented instead.

Browser example

See `example.html`.

Permalinks

Version 8.0.0 completely reworked the way permalinks work in order to offer more accessible options out of the box. You can also make your own permalink.

Instead of a single default way of rendering permalinks (which used to have a poor UX on screen readers), we now have multiple styles of permalinks for you to chose from.

```
const anchor = require('markdown-it-
        anchor')
const md = require('markdown-it')()

md.use(anchor, {
  permalink:
        anchor.permalink[styleOfPermalink]
        (permalinkOpts)
})
```

Here, `styleOfPermalink` is one of the available styles documented below, and `permalinkOpts` is an options object.

All renderers share a common set of options:

Name	Description	Default
`class`	The class of the permalink anchor.	`header-anchor`
`symbol`	The symbol in the permalink anchor.	#
`renderHref`	A custom permalink `href` rendering function.	See permalink.js
`renderAttrs`		


```
    assistiveText: title => `Permalink
        to “${title}”`,
    visuallyHiddenClass: 'visually-
        hidden',
    wrapper: ['', '']
  })
})
```

```
<h2 id="title">Title</h2>
<a class="header-anchor"
      href="#title">
  <span class="visually-
      hidden">Permalink to “Title”</
      span>
  <span aria-hidden="true">#</span>
</a>
```

By using a visually hidden element for the assistive text, we make sure that the assistive text can be picked up by translation services, as most of the popular translation services (including Google Translate) currently ignore `aria-label`.

If you prefer an alternative method for the assistive text, see other styles:

`aria-label` variant

This removes the need from a visually hidden `span`, but will likely hurt the permalink experience when using a screen reader through a translation service.

```
const anchor = require('markdown-it-
        anchor')
const md = require('markdown-it')()
```

Name	Description	Default
`visuallyHiddenClass`	The class you use to make an element visually hidden.	`undefined`, required for `visually-hidden` style
`space`	Add a space between the assistive text and the permalink symbol.	`true`
`placement`	Placement of the permalink symbol relative to the assistive text, can be `before` or `after` the header.	`after`
`wrapper`	Opening and closing wrapper string, e.g. `['', '']`. See common options.	`null`

```
const anchor = require('markdown-it-
        anchor')
const md = require('markdown-it')()

md.use(anchor, {
  permalink:
        anchor.permalink.linkAfterHeader({
    style: 'visually-hidden',
```


Name	Description	Default
	A custom permalink attributes rendering function.	See `permalink.js`

For the `symbol`, you may want to use the link symbol, or a symbol from your favorite web font.

Header link

This style wraps the header itself in an anchor link. It doesn't use the `symbol` option as there's no symbol needed in the markup (though you could add it with CSS using `::before` if you like).

It's so simple it doesn't have any behaviour to custom, and it's also accessible out of the box without any further configuration, hence it doesn't have other options than the common ones described above.

You can find this style on the MDN as well as HTTP Archive and their Web Almanac, which to me is a good sign that this is a thoughtful way of implementing permalinks. This is also the style that I chose for my own blog.

Name	Description	Default
`safariReaderFix`	Add a `span` inside the link so Safari shows headings in reader view. See common options.	`false` (for backwards compatibility)

```
const anchor = require('markdown-it-
        anchor')
const md = require('markdown-it')()

md.use(anchor, {
  permalink:
        anchor.permalink.headerLink()
})
```

```
<h2 id="title"><a class="header-anchor"
        href="#title">Title</a></h2>
```

The main caveat of this approach is that you can't include links inside headers. If you do, consider the other styles.

Also note that this pattern breaks reader mode in Safari, an issue you can also notice on the referenced websites above. This was already reported to Apple but their bug tracker is not public. In the meantime, a fix mentioned in the article above is to insert a `span` inside the link. You can use the `safariReaderFix` option to enable it.

```
const anchor = require('markdown-it-
        anchor')
const md = require('markdown-it')()

md.use(anchor, {
  permalink:
        anchor.permalink.headerLink({
        safariReaderFix: true })
})
```



```
<h2 id="title"><a class="header-anchor"
        href="#title"><span>Title</
        span></a></h2>
```

Link after header

If you want to customize further the screen reader experience of your permalinks, this style gives you much more freedom than the header link.

It works by leaving the header itself alone, and adding the permalink *after* it, giving you different methods of customizing the assistive text. It makes the permalink symbol `aria-hidden` to not pollute the experience, and leverages a `visuallyHiddenClass` to hide the assistive text from the visual experience.

Name	Description	Default
`style`	The (sub) style of link, one of `visually-hidden`, `aria-label`, `aria-describedby` or `aria-labelledby`.	`visually-hidden`
`assistiveText`	A function that takes the title and returns the assistive text.	`undefined`, required for `visually-hidden` and `aria-label` styles