# Num Slider

webtool.one

## Source files

| File name | Description |
| --- | --- |
| `main.js` | The component core, it contains the `NumSlider` class. It should be used to import and use a slider. |
| `options.js` | Options. It also includes parameters for internal use. Use it for parameter reference. |
| `num-slider.vue` | Vue component. It can be used directly in the Vue environment. |
| `src/accessors.js` | Getters and setters. |
| `src/actions.js` | Actions, e. g. moving, blocking, etc. |
| `src/attrs.js` | Getter `NumSlider.attrs`: it returns HTML attributes. |
| `src/constants.js` | Constants for calculating dimensions, SVG coordinates, etc. |
| `src/events.js` | Event functionality. |
| `src/helpers.js` | Helper functions. |
| `src/init-check.js` | Initial checks. |
| `src/ops.js` | Operations, e. g. setting nums, validation, etc. |
| `src/validate-obj.js` | Helpers for validating objects. |
| `src/svghtml.vue` | Vue component for icon rendering. |
| `src/vue-directives.js` | Vue directives. |

## Architecture

### Isolated logic

This functionality is made to be almost framework/presentation layer agnostic. This means that all logic is independent of HTML rendering, which includes the following features:

- HTML attributes (dynamic element classes, IDs, HTML Symbols, etc.) are separated from the presentation layer and provided to it on each update (when `NumSlider` state changes);
- To react to HTML updates standard JavaScript Observer API is used;
- To update the presentation layer after respective state changes, the special method `NumSlider.update` is used. After all, such an approach gives strict control over when display layer update should be performed theoretically providing more performant execution (touch HTML only when all necessary operations are done). Also, the resulting code should be less error-prone;
- Code is divided into separate blocks: operations, HTML attributes, constants, actions, etc.

Due to these points to use the slider in any project you need to:

1. Create a presentation, i. e. HTML, layer: it can be any reactive JavaScript framework or even plain HTML with vanilla JavaScript;
2. Connect presentation layer with logic, i. e. pass state to instantiated `NumSlider` instance and update HTML according to received result.

If you don't mind using Vue.js, it's already implemented for the presentation layer and slider is ready for immediate usage.

If you're adapting the presentation layer for some particular framework, please check `.vue` template files to create proper HTML elements.

Parameters for internal use, i. e. directly in the logic, are separated for convenience: check it also for satisfactory values.

For the styling guide please see the appropriate section of this docs. Please be aware that in HTML there are some elements (not so much so far) that have hard-coded attributes: these should be consistent with styles and `NumSlider.attrs` getter (source file `src/attrs.js`).

### HTML elements

In general, HTML elements are intended for internal use only. However, in case you need them, all HTML elements are accessible via `NumSlider.refs.<element name>`, e. g. `DTPicker.refs.root`. For reference please see source files. Also please note, that some of these `DTPicker.refs` might be refreshed after the slider gets updated.

### Classes (prototypes)

#### NumSlider

In Vue.js implementation it's directly accessible from outside the slider as component `ref`, e. g. in your component use: `<imported slider component name>.value.slider`. The slider instance is exposed via `defineExpose`, so to access the slider instance you need to get the component containing it first.

To use `NumSlider` as a standalone object you need to instantiate it and provide proper HTML for the representation layer. The most obvious way to do it is to port HTML from the Vue component, i. e. `num-slider.vue`.

Please refer to the source files for details.

## Usage

Slider can be used in 2 ways:

1. As a Vue component;
2. As a standalone functionality.

To use slider as a Vue component:

```
<template>
    <!-- NOTE: update model on event `update:model`, v-model will not work! -->
    <NumSlider v-bind='options' @update:model='model = $event' />
</template>

<script setup>
import { ref } from 'vue'
import NumSlider from 'num-slider/js/num-slider'

/* Options */
const options = ref({
    id: 'slider-id',
    minMax: [0, 100],
    model: [10, 40] /* Initial data */
    /* ... other options ... */
})

 /* Data with initial values, will be updated. */
const model = ref([10, 40])

/* ... do something with `model` ... */
</script>
```

To use slider as a standalone functionality:

```
import NumSlider from 'num-slider/main'

/* Create state object.
   Reference to it shoud be passed to NumSlider constructor.
   It is updated after each slider operation.
   Object properties depend on your approach,
   see `num-slider/num-slider.vue for details`
*/
const state = { ... }

const options = { ... }

/* Instantiate NumSlider instance: pass options and reference to state. */
const slider = new NumSlider(options, state)

/* ... react to `state` changes and update HTML ... */
```

Please note that to use the slider as a standalone functionality, you need to implement a presentation layer (HTML) for it. See Vue component `num-slider.vue` for example.

## Options

| Option | Type | Default | Description |
|---|---|---|---|
| `minMax` | `Array` | `null` | Min and max of slider range. REQUIRED. If not provided, stub will be used. |
| `model` | `Number \| Array` | `null` | Data model of slider values. REQUIRED. If not provided, stub will be used. |
| `id` | `String` | `'num-slider'` | ID of container. |
| `blocked` | `Boolean` | `false` | Whether to block (disable) entire component. |
| `blockElement` | `Element \| Document \| Window` | `null` | If slider's SVG will intersect this element box, then component will be blocked. |
| `vertical` | `Boolean` | `false` | Whether to use vertical alignment. |
| `step` | `Number` | `1` | Step to change values. |
| `numAxisScale` | `Boolean` | `true` | Whether to scale nums display, i. e. 1 thousand -> 1K, 1 million -> 1M. |
| `inputsOnTop` | `Boolean` | `true` | Whether to display inputs & control icons on top (on right for vertical orientation). |
| `numsOnTop` | `Boolean` | `false` | Whether to display slider nums on top. If vertical, on right. |
| `displayInputs` | `Boolean` | `true` | Whether to display inputs. |
| `barControls` | `Boolean` | `true` | Whether to display slider controls, i. e. circles. |
| `disableFormInputs` | `Boolean \| Function` | `false` | Whether to disable form inputs. Function: format selected number, implies disabled inputs. |
| `displayControlIcons` | `Boolean` | `true` | Whether to display arrows 'to-start' and 'to-end'. |
| `useRevert` | `Boolean` | `true` | Whether to use revert functionality. |
| `numsInterval` | `Number \| Array \| Boolean \| Function` | `true` | The number of steps between nums displayed. `0 \| null \| false` -> don't display. `Array` -> points to display. Function: use function w/ signature `(<min>, <max>, <step>)`. `true` -> auto. |
| `ticksInterval` | `Number \| Array \| Boolean \| Function` | `true` | Analogous to `numsInterval`. |
| `formatNum` | `Function` | `null` | Function to format displayed number, one argument - number to be formatted. |
| `forceShowEdgeTicks` | `Boolean` | `false` | In some cases due to interval start or end ticks may be omitted. Whether to force display of them. |
| `forceShowEdgeNums` | `Boolean` | `false` | Analogous to `forceShowEdgeTicks`. |
| `stickyMode` | `Boolean` | `true` | Whether to animate control(s) auto-move to step point. |
| `clickMode` | `Boolean` | `true` | Whether to set value on click on bar. |

| Option | Type | Default | Description |
|---|---|---|---|
| `clickModeSVG` | `Boolean` | `true` | Whether to set value on click on entire SVG element. It has no effect if `clickMode = false`. |
| `animation` | `Object` | `{ duration: 250, easing: 'ease-out' }` | Slide animation on number input or click on bar. |
| `animationStick` | `Object` | `{ duration: 50, easing: 'ease-out' }` | Animation of sticking to valid number in case mouse button released on step span. |
| `inputUpdateDelayMS` | `Number` | `400` | Delay of updating slider after number input. |
| `paramsSVG` | `Object` | `{}` | Params for SVG. See below. |

All animation options comply with the Web Animation API.

## SVG parameters

All parameters are numbers with one exception: `alignEdgeNums` is Boolean. For particular values please see file `options.js`.

| Option | Description |
|---|---|
| `viewBox` | SVG `viewBox` attribute. |
| `barShift` | Additional shift of slider bar. Relative to `viewBox[3]` (`viewBox[2]` if vertical). |
| `lineWeightK` | The line thickness. Relative to `viewBox` height (width if vertical). |
| `lineRk` | Bar line round corners factor. Relative to bar line weight. |
| `controlRk` | Control radius factor. Relative to bar line weight. |
| `tickWeightK` | Tick height factor. Relative to bar line weight. |
| `ticksShift` | Ticks shift along Y axis (X if vertical). Relative to bar line weight and bar line weight center. |
| `numOffset` | Number offset. Relative to bar line weight. |
| `numSizeK` | Number size factor. Relative to bar line weight. |
| `numScalePadK` | Margin of scale text (e. g. `K` for thousand). Relative to `numSize`. |
| `controlStrokeK` | Control stroke width factor. Relative to control Diameter. |
| `alignEdgeNums` | Whether to align start and end nums to the slider edge instead of tick. |

Please note that some options depend on each other, so changing one parameter might require changing the other. For example, slider controls (circles) might be enlarged to the point when overlapping numbers. Consequently, when setting any parameter, please ensure that all sizes are properly balanced.

In addition, there are some options for internal use. Please see source file `options.js` for details.

## Actions

Besides the core actions (values selection), the are additional ones:

1. **Blocking**. Component will be blocked when option `blocked` is set to `true`. Additionally, blocking and unblocking can be done via calling respective methods: `NumSlider.block()` and `NumSlider.unblock()`.
2. **Setting stub**. Stub will be set when no required options (`minMax` and `model`) are provided. In other cases stub can be set/removed via setter/getter `NumSlider.stub`.

### Blocking on changing visibility

When only a part of or entire slider is not visible, there is no sense to maintain its activity. So in such cases, a slider will be blocked. The main reason for such invisibility is scrolling, so the element on which the scroll event slider will check its visibility should be set via the option `blockElement`. If no element is provided, then the `window` object will be used.

### Setting stub on initial state

When the stub is set on the initial state then it will always display only one input (if option `displayInputs = true`) because there is no knowledge about whether this is a range or not. Consequently, if we have data in the component, then the stub will respect it and will display the proper number of inputs.

### A note on SVG coordinates

The slider should be aware of proper screen coordinates to perform sliding and other actions. For this reason, any changes in page layout or sizes might lead to the wrong positioning of SVG points (`viewBox`).

To cope with such situations, the SVG coordinates matrix (`SVGMatrix` or `DOMMatrix`) gets updated when clicking on the slider control or a control starts moving. However, there is a chance that for some edge cases this will be not enough. So if you have some problems with SVG positioning, please ensure the SVG matrix is properly set. This can be done via `NumSlider.CTM` property, i. e. refresh it when needed: `<instance>.CTM = <instance>.refs.svg.getScreenCTM()`.

## Events

### Emits

All events have names constructed from slider ID plus event name: `<slider id>:<event name>`.

| Event name | Description | Payload |
|---|---|---|
| `update:model` | Emitted when selected data changes. | Selected data. |

### Listeners

All listeners are intended for internal use only. Please refer to source files, especially, `src/events.js`, for details.

## Styling

Styling is done with beautiful CSS preprocessor Stylus. So you need to use it too in case comprehensive control is needed. Otherwise, simply import precompiled CSS from `num-slider/css/num-slider.css` but be aware that in this case picker will use default CSS values for all its instances.

When using Stylus, to overwrite some or all of the theme variables, import main theme and styles as follows (there may be more than one slider instance on one page):

```
/* Theme ID must be set in case of several slider elements. */
NumSliderID = 'slider-id'

/* import main theme to set or overwrite all variables to defaults. */
@import 'num-slider/css/src/theme'

/* Overwrite variables: colors, etc. */
colorNumSliderFG = yellowgreen

/* import styles to be overwritten. */
@import 'num-slider/css/src/main'
```

Using media queries:

```
NumSliderID = 'slider-id'
@import 'num-slider/css/src/theme'
@import 'num-slider/css/src/main'

/* Change sizes for viewport width <= 500px */
@media (max-width: 500px)
    NumSliderID = 'slider-id'  /* Specify slider ID. */
    inputsW = 4em /* Set variables. */
    @import 'num-slider/css/src/main' /* Import styles. */
```

When setting options of `paramsSVG` please ensure that all values are properly balanced.

> Changing the displayed number color on the `hover` event is intentionally disabled: clicking on a number is actually clicking on slider SVG to produce a precise value, so it may be misleading. When clicking on the designated number, the user will expect that this particular value will be selected and in this regard, we can lose the ability to select values that are very close to displayed numbers.

When using Vue and applying its transition build-in component, it may not work as expected due to the self-updating nature of `NumSlider.attrs`. In such cases wrap the slider component in `div` and apply transition on it, like so:

```
<template>
    <transition name='fade' mode='out-in'>
        <div class='slider-wrapper' :key='sliderKey'>
            <NumSlider v-bind='options' @update:model='model = $event' />
        </div>
    </transition>
</template>

<script setup>
    import {  computed } from 'vue'
    /* ... some code ... */
    const sliderKey = computed( _ => ... )
    /* ... some code ... */
</script>
```

All icons live in the `icons` directory. To change existing ones, simply replace icon files.

---

Web version

Demo