GitHub | This repository Search | Explore Features Enterprise Pricing | Sign up | Sign in

bevacqua / **rome**

Watch 73 | Star 2,308 | Fork 164

Code | Issues 27 | Pull requests 13 | Pulse | Graphs

Customizable date (and time) picker. Opt-in UI, no jQuery! https://bevacqua.github.io/rome

298 commits | **3** branches | **93** releases | **10** contributors

Branch: **master** | New pull request | New file | Find file | HTTPS | https://github.com/bevacq | Download ZIP

bevacqua Release v2.1.22 — Latest commit 44e3d5f on Jul 23, 2015

dist	Release v2.1.22	7 months ago
example	Update example.js	2 years ago
resources	fix screenshot	2 years ago
src	This should probably be on another line now i've made the function to…	7 months ago
.editorconfig	Release v0.1.44	2 years ago
.gitignore	Ignore .idea folder	a year ago
.jshintignore	rm start-up need for .on('ready', fn)	a year ago
.jshintrc	rm start-up need for .on('ready', fn)	a year ago
.npmignore	add screen capture	2 years ago
CHANGELOG.md	update changelog	7 months ago
LICENSE	Release v0.1.44	2 years ago
README.md	Emit events 'back' and 'next' with \`month\` arg	10 months ago
bower.json	Release v2.1.22	7 months ago
gulpfile.js	Restored 'use-strict', did not intend to remove that. I renamed the '…	7 months ago
index.html	start supporting multi-month calendar	2 years ago
package.json	Release v2.1.22	7 months ago

README.md

rome

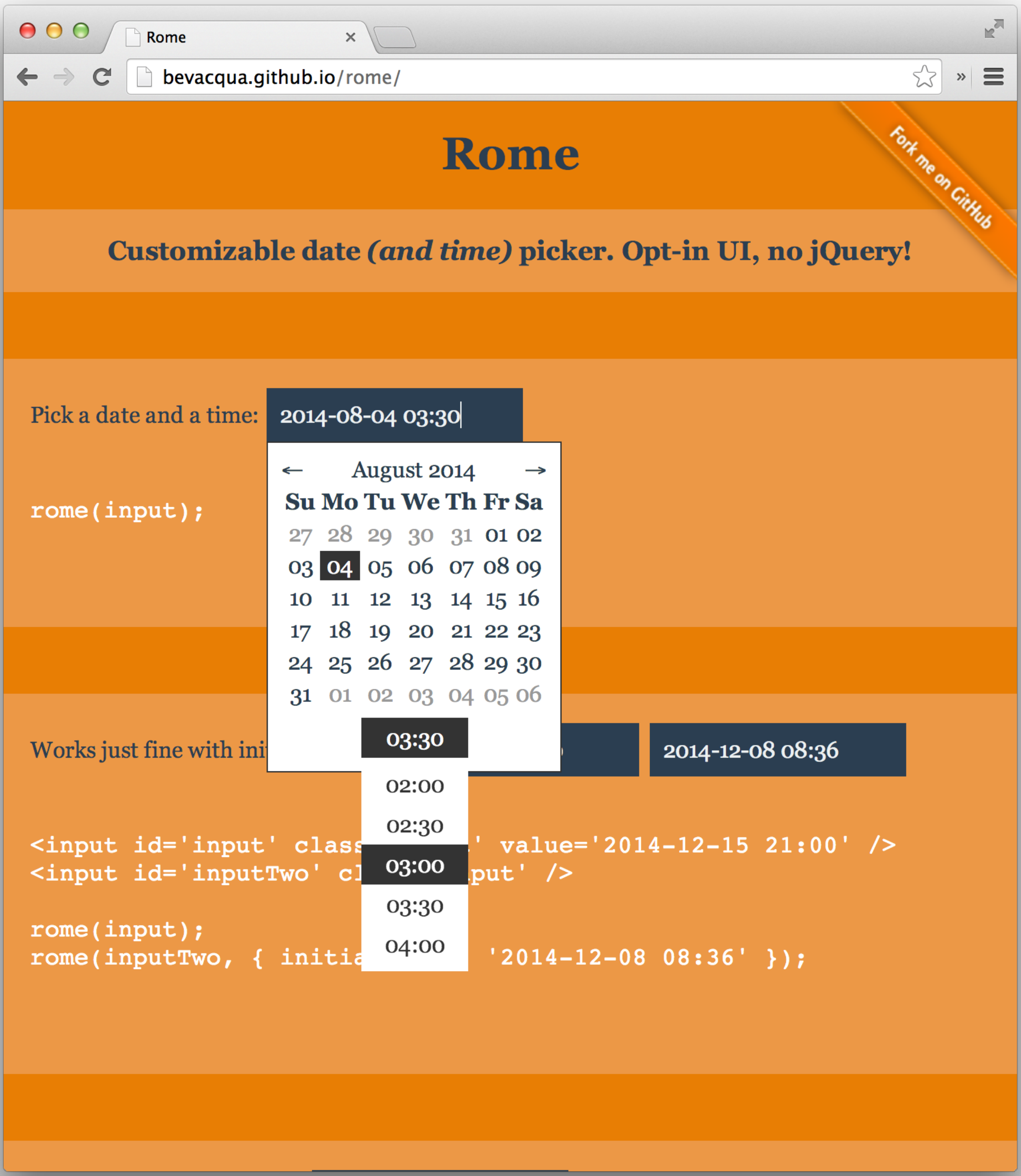
Gittip | Flattr

> Customizable date *(and time)* picker. Opt-in UI, no jQuery!

Rome wasn't built in a day. Browser support includes every sane browser and **IE7+**.

Demo!

You can see a live demo here.



```
rome(input);
```

```
<input id='input' class' value='2014-12-15 21:00' />
<input id='inputTwo' cl put' />

rome(input);
rome(inputTwo, { initia '2014-12-08 08:36' });
```

Oh, `rome` synchronizes in real-time with inputs, never steals focus, and its CSS is entirely customizable!

Rome depends on `moment`. It doesn't depend on jQuery or other weird frameworks, though.

Install

From npm or Bower.

```
npm install --save rome
```

```
bower install --save rome
```

Note that if you're using the standalone version, the API is published under the `rome` global. If you're using CJS, then you'll have to `require('rome')`.

Setup

You can use your own distribution of `moment`, using `rome.standalone.js`.

```
<script src='moment.js'></script>
<script src='rome.standalone.js'></script>
```

You could just use the bundled `rome.js` distribution, which comes with `moment` in it.

```
<script src='rome.js'></script>
```

If you need to do anything regarding internationalization, refer to `moment` for that. Ideally, make those changes before starting to create Rome calendar components.

API

The API in `rome` exposes a few properties.

`rome.find(elem)`

If a calendar is associated to the provided `elem`, then that calendar is returned, otherwise returns `null`. DOM elements can only have one associated calendar.

`rome(elem, options={})`

This method creates a calendar instance and associates it to the provided `elem`. This association can't be undone even by `.destroy()` ing the `rome` instance, because it can be `.restore()` d later. Subsequent calls to `rome(elem)` will return the associated calendar, instead of creating a new one (see `rome.find(elem)`). Think of this as a *"caching feature"*.

Creating a calendar has a ton of options. These have reasonable defaults that are easy to adjust, too. The options are listed below.

Option	Description
`appendTo`	DOM element where the calendar will be appended to. Takes `'parent'` as the parent element
`autoClose`	When set to `true`, the calendar is auto-closed when picking a day _(or a time if `time: true` and `date: false`). A value of `'time'` will only auto-close the calendar when a time is picked.
`autoHideOnBlur`	Hides the calendar when focusing something other than the input field
`autoHideOnClick`	Hides the calendar when clicking away
`date`	The calendar shows days and allows you to navigate between months
`dateValidator`	Function to validate that a given date is considered valid. Receives a native `Date` parameter.
`dayFormat`	Format string used to display days on the calendar
`initialValue`	Value used to initialize calendar. Takes `string`, `Date`, or `moment`
`inputFormat`	Format string used for the input field as well as the results of `rome`
`invalidate`	Ensures the date is valid when the field is blurred
`strictParse`	Compares input strictly against `inputFormat`, and partial matches are discarded
`max`	Disallow dates past `max`. Takes `string`, `Date`, or `moment`
`min`	Disallow dates before `min`. Takes `string`, `Date`, or `moment`
`monthFormat`	Format string used by the calendar to display months and their year
`monthsInCalendar`	How many months get rendered in the calendar

`required`	Is the field required or do you allow empty values?
`styles`	CSS classes applied to elements on the calendar
`time`	The calendar shows the current time and allows you to change it using a dropdown
`timeFormat`	Format string used to display the time on the calendar
`timeInterval`	Seconds between each option in the time dropdown
`timeValidator`	Function to validate that a given time is considered valid. Receives a native `Date` parameter.
`weekdayFormat`	Format used to display weekdays. Takes `min` *(Mo)*, `short` *(Mon)*, `long` *(Monday)*, or an array with seven strings of your choosing.
`weekStart`	Day considered the first of the week. Range: Sunday `0` - Saturday `6`

Note that in the case of input fields, when `initialValue` isn't provided the initial value is inferred from `elem.value` instead. In the case of inline calendars, `new Date()` will be used as a default if none is provided.

Inlining the Calendar

If you pass in an element other than an input tag, then this method behaves slightly differently. The difference is that `appendTo` becomes the provided `elem`, and the calendar won't attach itself to an input element. The options listed below will be ignored.

- `autoHideOnBlur`, because there is no input field that can be tracked for `blur` events
- `invalidate`, because there is no input field to keep consistent with the calendar component
- `required`, because you can easily do that on an input field
- `styles.positioned`, because the calendar will be considered inlined

All of the other options still apply, and identical behavior should be expected.

Default Options

If you don't set an option, the default will be used. You can look up the defaults here, or below.

```
{
  "appendTo": document.body,
  "autoClose": true,
  "autoHideOnBlur": true,
  "autoHideOnClick": true,
  "date": true,
  "dateValidator": Function.prototype,
  "dayFormat": "DD",
  "initialValue": null,
  "inputFormat": "YYYY-MM-DD HH:mm",
  "invalidate": true,
  "max": null,
  "min": null,
  "monthFormat": "MMMM YYYY",
  "monthsInCalendar": 1,
  "required": false,
  "strictParse": false,
  "styles": {
    "back": "rd-back",
    "container": "rd-container",
    "date": "rd-date",
    "dayBody": "rd-days-body",
    "dayBodyElem": "rd-day-body",
    "dayConcealed": "rd-day-concealed",
    "dayDisabled": "rd-day-disabled",
    "dayHead": "rd-days-head",
    "dayHeadElem": "rd-day-head",
    "dayRow": "rd-days-row",
    "dayTable": "rd-days",
    "month": "rd-month",
    "next": "rd-next",
    "positioned": "rd-container-attachment",
    "selectedDay": "rd-day-selected",
    "selectedTime": "rd-time-selected",
    "time": "rd-time",
    "timeList": "rd-time-list",
    "timeOption": "rd-time-option"
  },
  "time": true,
  "timeFormat": "HH:mm",
  "timeInterval": 1800,
  "timeValidator": Function.prototype,
  "weekdayFormat": "min",
  "weekStart": moment().weekday(0).day()
}
```

Rome API

When you create a calendar with `rome(elem)` , you'll get a `cal` instance back. This has a few API methods. Most of these methods return the calendar instance whenever possible, allowing for method chaining.

`.show()`

Shows the calendar. If associated with an input, the calendar gets absolutely position right below the input field.

`.hide()`

Hides the calendar.

`.id`

Auto-generated unique identifier assigned to this instance of Rome.

`.container`

The DOM element that contains the calendar.

`.associated`

The associated DOM element assigned to this calendar instance. This is the input field or parent element that you used to create the calendar.

`.getDate()`

Returns the current date, as defined by the calendar, in a native `Date` object. If `required: false` you'll get `null` when the input field is empty.

`.getDateString(format?)`

Returns the current date, as defined by the calendar, using the provided `options.inputFormat` format string or a format of your choosing. If `required: false` you'll get `null` when the input field is empty.

`.getMoment()`

Returns a copy of the `moment` object underlying the current date in the calendar. If `required: false` you'll get `null` when the input field is empty.

`.destroy()`

Removes the calendar from the DOM and all of its associated DOM event listeners. The only responsive API method becomes the `.restore` method described below, the rest of the API becomes no-op methods. After emitting the `destroyed` event, all event listeners are removed from the instance.

`.destroyed`

Returns `true` when the calendar is in a destroyed state and `false` otherwise.

`.restore(options?)`

Restores the calendar, using the provided options (or the default options). The associated DOM element can't be changed. The API methods are restored to their original functionality.

`.options(options?)`

If an options object is provided, it destroys the calendar and initializes it with the provided options. Effectively the same as calling `.restore(options)` immediately after calling `.destroy()` .

If no options object is provided, a copy of the current options is returned.

`.options.reset()`

Resets the options to the factory defaults. Effectively the same as calling `.options({})` while preserving the `appendTo` option.

`.emitValues()`

Emits all of the data events listed below. Mostly used internally, **should be avoided** in consumer-land.

`.setValue(value)`

Sets the current date to the provided `value` , but only if that value is valid according to the rules defined by the calendar. Takes `string` , `Date` , or `moment` . Mostly used internally, and it doesn't emit any events.

`.refresh()`

Forces a refresh of the calendar. This method will redraw the month and update the dates that can be selected in accordance with `dateValidator` and `timeValidator` .

`.back()`

Steps the calendar display back by one month. Equivalent to clicking the 'back' button. Returns `undefined` .

`.next()`

Steps the calendar display forward by one month. Equivalent to clicking the 'next' button. Returns `undefined` .

Events

Rome calendars also provide a few events you can subscribe to. These events are published through an event emitter created using `contra` . These events are listed below.

Event	Arguments	Description
`ready`	`[options]`	The calendar has been `.restore` d
`destroyed`	`[]`	The calendar has been `.destroy` ed
`data`	`[value]`	The date may have been updated by the calendar. Value of `.getDateString()` is provided
`year`	`[year]`	The year may have been updated by the calendar. Value of `moment.year()` is provided
`month`	`[month]`	The month may have been updated by the calendar. Value of `moment.month()` is provided
`day`	`[day]`	The day may have been updated by the calendar. Value of `moment.date()` is provided
`time`	`[time]`	The time may have been updated by the calendar. Formatted time string is provided
`show`	`[]`	The calendar has been displayed
`hide`	`[]`	The calendar has been hidden
`back`	`[month]`	The calendar view has been moved back a month to the value `moment.month()`
`next`	`[month]`	The calendar view has been moved forward a month to the value `moment.month()`

Date and Time Validator

Please note that `dateValidator` and `timeValidator` both receive a native `Date` object as a parameter. These methods are expected to return `undefined` or `true` if the date is deemed valid, and `false` in case the date is invalid. If `dateValidator` returns `false` , the validation process will try to find a valid date near the desired date.

If `dateValidator` passes for a given date, the `timeValidator` will attempt to validate that date as well. If the time is invalid, the day will be probed for a valid time. This validation starts at the desired time, and grows in `timeInterval` increments. When the end of the day is reached, validation resumes at the start of the day instead of leaping to the next day.

`rome.val`

There are a few default validator factories provided by Rome to make your life easier.

These methods take a `moment` , a `Date` , a `string` that can be parsed into a `moment` using `inputFormat` , or a DOM element that Rome could use to look up another Rome instance.

If you passed in a DOM element, the validator will look up the associated Rome instance and validate using its value. The first time the validator is executed on any inline calendar, the `'data'` event for that calendar will be hooked to refresh the related calendar.

For usage examples you can refer to the demos.

`rome.val.afterEq(value)`

Returns whether the date is after the provided value. The comparison uses `>=` , meaning it's inclusive.

`rome.val.after(value)`

Returns whether the date is after the provided value. The comparison uses `>` , meaning it's exclusive.

`rome.val.beforeEq(value)`

Returns whether the date is before the provided value. The comparison uses `<=` , meaning it's inclusive.

`rome.val.before(value)`

Returns whether the date is before the provided value. The comparison uses `<` , meaning it's exclusive.

`rome.val.except(left, right)`

Returns whether the date is any date except the provided value. You can provide a wide variety of input values. Keep in mind `Date` , `string` , `moment` , and the DOM element used to find another calendar are all valid input types.

Providing `left` only means "any date except this one"

If you use `rome.val.except('2014-08-09')` , then `'2014-08-09'` is invalid.

Providing `left` and `right` means "any date that's not in this range"

If you use `rome.val.except('2014-08-09', '2014-09-01')` , then anything between `'2014-08-09'` and `'2014-09-01'` is invalid.

If `left` is an array, each element in the array is treated as the simple case described above

In this case, `right` is completely ignored. Every item in the array is treated as follows.

If the item is single, then a rule is built on that single date

Using `rome.val.except(['2014-08-09', '2014-09-01'])` means that `'2014-08-09'` and `'2014-09-01'` are both invalid dates.

If the item is an array, the first two items are used to determine a date range

Using `rome.val.except([['2014-08-09', '2014-09-01']])` means anything between `'2014-08-09'` and `'2014-09-01'` is invalid.

These two types of entries can be combined in any way you like. Each entry will exclude additional dates.

For instance, `[['2014-04-05', '2014-04-15'], ['2014-04-25', '2014-04-30'], '2014-05-05']` means that April 05 to 15, and April 25 to 30, along with May 05 are all invalid dates.

`rome.val.only(left, right)`

Identical behavior to `rome.val.except` , except for the fact that the selected dates become **the only valid dates**, rather than the **only invalid dates**.

`rome.moment`

Exposes the `moment` instance used by Rome. To change the `moment` instance, refer to `rome.use(moment)` .

`rome.use(moment)`

Sets the instance of `moment` used by Rome.

Development

Start by installing any dependencies.

```
npm install
```

Then run the Gulp `watch` task.

```
gulp watch
```

Lastly open the page and any changes you make just need a browser refresh.

```
open index.html
```

License

MIT