
 This repository Search

Pull requestsIssuesGist



sachinchoolur / angular-trix

Watch 10Star 176Fork 17

[Code](#) [Issues 9](#) [Pull requests 2](#) [Wiki](#) [Pulse](#) [Graphs](#)

A rich WYSIWYG text editor directive for angularjs. <http://sachinchoolur.github.io/angular-trix/>

10 commits

2 branches

3 releases

1 contributor

Branch: master


New pull request

Create new file

Upload files

Find file

Clone or download



sachinchoolur

Update bower.json

Latest commit 3eee444 on 21 Dec 2015

|                 |                     |              |
|-----------------|---------------------|--------------|
| dist            | Initial commit      | 7 months ago |
| src             | Screenshot..        | 7 months ago |
| test            | Initial commit      | 7 months ago |
| .editorconfig   | Initial commit      | 7 months ago |
| .gitignore      | Initial commit      | 7 months ago |
| .jscsrc         | Initial commit      | 7 months ago |
| .jshintrc       | Initial commit      | 7 months ago |
| .travis.yml     | Initial commit      | 7 months ago |
| Gruntfile.js    | Initial commit      | 7 months ago |
| LICENSE         | Initial commit      | 7 months ago |
| README.md       | Update README.md    | 7 months ago |
| bower.json      | Update bower.json   | 7 months ago |
| contributing.md | Initial commit      | 7 months ago |
| package.json    | Update package.json | 7 months ago |

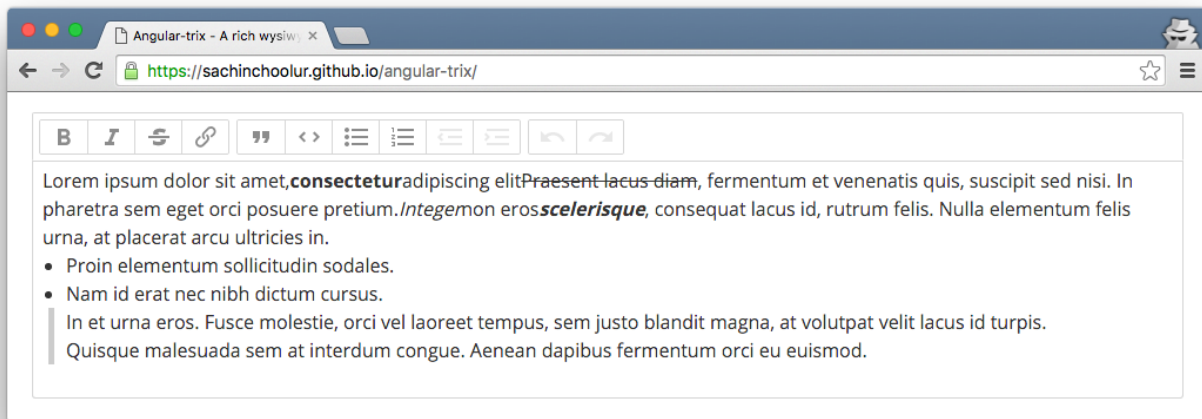
README.md

license MITbuild passingbower v1.0.2npm v1.0.2

angular-trix

A rich wysiwyg text editor directive for angularjs.

Angularjs directive for trix editor



## Demo

[Angular-trix demo](#). [Plunker demo](#)

## Install

You can get it on npm.

```
npm install angular-trix --save
```

Or bower, too.

```
bower install angular-trix --save
```

If you're not into package management, just [download a ZIP](#) file.

## Setup

**First, include angularjs, trix.js and trix.css into your document.**

```
<link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/trix/0.9.2/trix.css">

<script src="//ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/trix/0.9.2/trix.js"></script>
```

trix.css includes default styles for the Trix toolbar, editor, and attachments. Skip this file if you prefer to define these styles yourself.

To use your own polyfills, or to target only browsers that support all of the required standards, include trix-core.js instead.

**Then Include angular-trix.js.**

```
<script src="dist/angular-trix.min.js"></script>
```

Add `angularTrix` dependency to your module

```
var myApp = angular.module('app', ['angularTrix']);
```

## Create trix-editor

Place an empty `<trix-editor></trix-editor>` tag on the page. Trix will automatically insert a separate `<trix-toolbar>` before the editor.

Like an HTML `<textarea>`, `<trix-editor>` accepts `autofocus` and `placeholder` attributes. Unlike a `<textarea>`, `<trix-editor>` automatically expands vertically to fit its contents.

Finally, add `angular-trix` directive and `ng-model` to the `<trix-editor></trix-editor>`.

```
<trix-editor angular-trix ng-model="foo"></trix-editor>
```

## Styling Formatted Content

To ensure what you see when you edit is what you see when you save, use a CSS class name to scope styles for Trix formatted content. Apply this class name to your `<trix-editor>` element, and to a containing element when you render stored Trix content for display in your application.

```
<trix-editor angular-trix ng-model="foo" class="trix-content"></trix-editor>
```

The default `trix.css` file includes styles for basic formatted content—including bulleted and numbered lists, code blocks, and block quotes—under the class name `trix-content`. We encourage you to use these styles as a starting point by copying them into your application's CSS with a different class name.

## Observing Editor Changes

The `<trix-editor>` element emits several events which you can use to observe and respond to changes in editor state.

- `trix-initialize` fires when the `<trix-editor>` element is attached to the DOM and its `editor` object is ready for use.
- `trix-change` fires whenever the editor's contents have changed.
- `trix-selection-change` fires any time the selected range changes in the editor.
- `trix-focus` and `trix-blur` fire when the editor gains or loses focus, respectively.
- `trix-file-accept` fires when a file is dropped or inserted into the editor. You can access the DOM `File` object through the `file` property on the event.
- `trix-attachment-add` fires after an attachment is added to the document. You can access the Trix attachment object through the `attachment` property on the event. If the `attachment` object has a `file` property, you should store this file remotely and set the attachment's URL attribute.
- `trix-attachment-remove` fires when an attachment is removed from the document. You can access the Trix attachment object through the `attachment` property on the event. You may wish to use this event to clean up remotely stored files.

You can use the following attributes to listen the trix events and implement your custom logic. You can access trix event and editor instance parameters in controller.

- `trix-initialize`
- `trix-change`
- `trix-selection-change`
- `trix-focus`

- `trix-blur`
- `trix-file-accept`
- `trix-attachment-add`
- `trix-attachment-remove`

```
<trix-editor ng-model="trix" angular-trix trix-initialize="trixInitialize(e, editor);" trix-change="trixChange(e, edi
```

```
// You can still access the trix event
var events = ['trixInitialize', 'trixChange', 'trixSelectionChange', 'trixFocus', 'trixBlur', 'trixFileAccept', 'trix

for (var i = 0; i < events.length; i++) {
  $scope[events[i]] = function(e, editor) {
    console.info('Event type:', e.type);
  }
};
```

For a live demonstration, open this [site](#) and just your console :)

## Storing Attached Files

Trix automatically accepts files dragged or pasted into an editor and inserts them as attachments in the document. Each attachment is considered *pending* until you store it remotely and provide Trix with a permanent URL.

To store attachments, listen for the `trix-attachment-add` event. Upload the attached files with XMLHttpRequest yourself and set the attachment's URL attribute upon completion. See the [Plunker](#) for detailed information.

If you don't want to accept dropped or pasted files, add `prevent-trix-file-accept = "true"` attribute to the trix editor.

## Editing Text Programmatically

You can manipulate a Trix editor programmatically through the `Trix.Editor` interface, available on each `<trix-editor>` element through its `editor` property. You can access editor property in controller via trix event parameter.

```
<trix-editor angular-trix ng-model="foo" trix-initialize="trixInitialize(e, editor);"></trix-editor>
```

```
// Controller
// @e trix event
// @editor Trix.Editor instance
$scope.trixInitialize = function(e, editor) {}
```

## Understanding the Document Model

The formatted content of a Trix editor is known as a *document*, and is represented as an instance of the `Trix.Document` class. To get the editor's current document, use the `editor.getDocument` method.

```
$scope.trixInitialize = function(e, editor) {
  editor.getDocument() // is a Trix.Document instance
}
```

You can convert a document to an unformatted JavaScript string with the `document.toString` method.

```
$scope.trixInitialize = function(e, editor) {
  var document = editor.getDocument()
  document.toString() // is a JavaScript string
}
```

## Getting and Setting the Selection

Trix documents are structured as sequences of individually addressable characters. The index of one character in a document is called a *position*, and a start and end position together make up a *range*.

To get the editor's current selection, use the `editor.getSelectedRange` method, which returns a two-element array containing the start and end positions.

```
$scope.trixInitialize = function(e, editor) {
  editor.getSelectedRange() // [0, 0]
}
```

You can set the editor's current selection by passing a range array to the `editor.setSelectedRange` method.

```
$scope.trixInitialize = function(e, editor) {
  // Select the first character in the document
  editor.setSelectedRange([0, 1])
}
```

## Collapsed Selections

When the start and end positions of a range are equal, the range is said to be *collapsed*. In the editor, a collapsed selection appears as a blinking cursor rather than a highlighted span of text.

For convenience, the following calls to `setSelectedRange` are equivalent when working with collapsed selections:

```
$scope.trixInitialize = function(e, editor) {
  editor.setSelectedRange(1)
  editor.setSelectedRange([1])
  editor.setSelectedRange([1, 1])
}
```

## Directional Movement

To programmatically move the cursor or selection through the document, call the `editor.moveCursorInDirection` or `editor.expandSelectionInDirection` methods with a *direction* argument. The direction can be either `"forward"` or `"backward"`.

```
$scope.trixInitialize = function(e, editor) {
  // Move the cursor backward one character
  editor.moveCursorInDirection("backward")

  // Expand the end of the selection forward by one character
  editor.expandSelectionInDirection("forward")
}
```

## Converting Positions to Pixel Offsets

Sometimes you need to know the x and y coordinates of a character at a given position in the editor. For example, you might want to absolutely position a pop-up menu element below the editor's cursor.

Call the `editor.getClientRectAtPosition` method with a position argument to get a `DOMRect` instance representing the left and top offsets, width, and height of the character at the given position.

```
$scope.trixInitialize = function(e, editor) {  
  var rect = editor.getClientRectAtPosition(0)  
  [rect.left, rect.top] // [17, 49]  
}
```

## Inserting and Deleting Text

The editor interface provides methods for inserting, replacing, and deleting text at the current selection.

To insert or replace text, begin by setting the selected range, then call one of the insertion methods below. Trix will first remove any selected text, then insert the new text at the start position of the selected range.

### Inserting Plain Text

To insert unformatted text into the document, call the `editor.insertString` method.

```
$scope.trixInitialize = function(e, editor) {  
  
  // Insert "Hello" at the beginning of the document  
  editor.setSelectedRange([0, 0])  
  editor.insertString("Hello")  
}
```

### Inserting HTML

To insert HTML into the document, call the `editor.insertHTML` method. Trix will first convert the HTML into its internal document model. During this conversion, any formatting that cannot be represented in a Trix document will be lost.

```
$scope.trixInitialize = function(e, editor) {  
  
  // Insert a bold "Hello" at the beginning of the document  
  editor.setSelectedRange([0, 0])  
  editor.insertHTML("<strong>Hello</strong>")  
}
```

### Inserting a File

To insert a DOM `File` object into the document, call the `editor.insertFile` method. Trix will insert a pending attachment for the file as if you had dragged and dropped it onto the editor.

```
$scope.trixInitialize = function(e, editor) {  
  
  // Insert the selected file from the first file input element  
  var file = document.querySelector("input[type=file]").file  
  editor.insertFile(file)  
}
```

### Inserting a Line Break

To insert a line break, call the `editor.insertLineBreak` method, which is functionally equivalent to pressing the return key.

```
$scope.trixInitialize = function(e, editor) {
```

```
// Insert "Hello\n"
editor.insertString("Hello")
editor.insertLineBreak()
}
```

## Deleting Text

If the current selection is collapsed, you can simulate deleting text before or after the cursor with the `editor.deleteInDirection` method.

```
$scope.trixInitialize = function(e, editor) {

  // "Backspace" the first character in the document
  editor.setSelectedRange([1, 1])
  editor.deleteInDirection("backward")

  // Delete the second character in the document
  editor.setSelectedRange([1, 1])
  editor.deleteInDirection("forward")
}
```

To delete a range of text, first set the selected range, then call `editor.deleteInDirection` with either direction as the argument.

```
$scope.trixInitialize = function(e, editor) {

  // Delete the first five characters
  editor.setSelectedRange([0, 4])
  editor.deleteInDirection("forward")
}
```

## Working With Attributes and Indentation

Trix represents formatting as sets of *attributes* applied across ranges of a document.

By default, Trix supports the inline attributes `bold`, `italic`, `href`, and `strike`, and the block-level attributes `quote`, `code`, `bullet`, and `number`.

## Applying Formatting

To apply formatting to the current selection, use the `editor.activateAttribute` method.

```
$scope.trixInitialize = function(e, editor) {
  editor.insertString("Hello")
  editor.setSelectedRange([0, 5])
  editor.activateAttribute("bold")
}
```

To set the `href` attribute, pass a URL as the second argument to `editor.activateAttribute`.

```
$scope.trixInitialize = function(e, editor) {
  editor.insertString("Trix")
  editor.setSelectedRange([0, 4])
  editor.activateAttribute("href", "http://trix-editor.org/")
}
```

## Removing Formatting

Use the `editor.deactivateAttribute` method to remove formatting from a selection.

```
$scope.trixInitialize = function(e, editor) {  
  editor.setSelectedRange([2, 4])  
  editor.deactivateAttribute("bold")  
}
```

## Formatting With a Collapsed Selection

If you activate or deactivate attributes when the selection is collapsed, your formatting changes will apply to the text inserted by any subsequent calls to `editor.insertString`.

```
$scope.trixInitialize = function(e, editor) {  
  editor.activateAttribute("italic")  
  editor.insertString("This is italic")  
}
```

## Adjusting the Indentation Level

To adjust the indentation level of block-level attributes, call the `editor.increaseIndentationLevel` and `editor.decreaseIndentationLevel` methods.

```
$scope.trixInitialize = function(e, editor) {  
  editor.activateAttribute("quote")  
  editor.increaseIndentationLevel()  
  editor.decreaseIndentationLevel()  
}
```

## Using Undo and Redo

Trix editors support unlimited undo and redo. Successive typing and formatting changes are consolidated together at five-second intervals; all other input changes are recorded individually in undo history.

Call the `editor.undo` and `editor.redo` methods to perform an undo or redo operation.

```
$scope.trixInitialize = function(e, editor) {  
  editor.undo()  
  editor.redo()  
}
```

Changes you make through the editor interface will not automatically record undo entries. You can save your own undo entries by calling the `editor.recordUndoEntry` method with a description argument.

```
$scope.trixInitialize = function(e, editor) {  
  editor.insertString("Hello")  
  editor.recordUndoEntry("Insert Text")  
}
```

## License

MIT License



