formtools Documentation

Release 1.0

nyxgear

Oct 04, 2019

Contents

1 What is Formtools? 1

2 How to include it 3

3 Dependencies 5

3.1 Define validators 5
3.2 How to validate 8
3.3 Settings 9
3.4 Changelog 13

CHAPTER 1

What is Formtools?

Formtools is a lightweight and powerful JQuery plugin that lets you validate, reset, and fill your HTML forms. It is designed to be extensible and customizable to fit your needs. You can easily define your input validators in the HTML markup and validate the form via Javascript with just one line of code!

CHAPTER 2

How to include it

Just put the following line before your javascript

```
<script src="https://cdn.jsdelivr.net/gh/nyxgear/formtools@v1.0/dist/
↪formtools.min.js"></script>
```

or, if you want to get formtools part of your project, you can include its minified file directly.

CHAPTER 3

Dependencies

-
-
- ()

Since formtools is a jQuery plugin it requires .
The only one real dependency is this!!

If you want validate dates you must also include

Formtools is thought to work by default on based template structure.
But! If you don't use bootstrap is not a problem.
You can easly define your custom settings to match your own template structure.
The only two things you need to set up are the *custom error class* and the *parent error class*.

3.1 Define validators

Formtools is designed to accept configurations and validators hierarchically.

Levels importance (from lowest to highest):

1. *Javascript configurations*
2. *Validators on the form tag* (They overwrite *Javascript configurations*)
3. *Validators on the input tag* (They overwrite both *Validators on the form tag* and *Javascript configurations*)

Validators are defined within the HTML form structure.

3.1.1 Validators on the input tag

You can put the following attributes on form inputs

<input [validators] ...>	
Required	`data-ft-required`
Optional	`data-ft-optional`
Minlength	`data-ft-min-length="5"`
Regex	`data-ft-regex="([A-Z])\w+"`
DateFormat	`data-ft-date-format="YYYY"`
DateRange	`data-ft-date-range="1900-01-01:0"`

Required

`data-ft-required`

It makes the field required, not empty.
Blanks are considered empty.

```
<input type="text" name="address" data-ft-required />
```

To overwrite the preset value: `data-ft-required="[true|false]"`

Optional

`data-ft-optional`

Use the optional validator to allow empty field and at the same time validate it if any content has been inserted.

```
<input type="text" name="address" data-ft-optional data-ft-regex="([A-Z])\w+" />
```

To overwrite the preset value: `data-ft-optional="[true|false]"`

Minlength

`data-ft-min-length="5"`

As the maxlength HTML input requirement, you can ask for a minimum length of the content.

```
<input type="text" name="address" data-ft-min-length="5" />
```

To overwrite the preset value just define the validator

Regex

`data-ft-regex="([A-Z])\w+"`

Just use a regex to validate the field.

```
<input type="text" name="address" data-ft-regex="([A-Z])\w+" />
```

To overwrite the preset value just define the validator

DateFormat

`data-ft-date-format="YYYY"`

Using you can specify a date format required.

```
<input type="date" name="costructionYear" data-ft-date-format="YYYY" />
```

To overwrite the preset value just define the validator

DateRange

`data-ft-date-range="1900-01-01:0"`

Based on the standard it expect a *StartDate:EndDate* sintax.

The 0 value means *no limit*.

WARNING!
The range must be defined in the ISO 8601 format, don't use the *data-ft-date-format* defined format!

The following example require a date after the first january of 1900

```
<input type="date" name="costructionDate" data-ft-date-range="1900-01-01:0" data-ft-
↪date-format="MM/DD/YYYY" />
```

To overwrite the preset value just define the validator

3.1.2 Validators on the form tag

You can put the these attributes on form tags

<form [validators] ...>		
Required	`data-ft-required`	All inputs in the form are required. It follows *Required* overwriting rules to overwrite *Javascript configured validators*
Optional	`data-ft-optional`	All inputs in the form are optional. It follows *Optional* overwriting rules to overwrite *Javascript configured validators*
Minlength	`data-ft-min-length="5"`	All inputs in the form must have at least 5 characters. It follows *Minlength* overwriting rules to overwrite *Javascript configured validators*
RegEx	`data-ft-regex="([A-Z])+"`	All inputs in the form must match the regex. It follows *Regex* overwriting rules to overwrite *Javascript configured validators*
DateFormat	`data-ft-date-format="YYYY"`	All "*type="date"*" inputs in the form must match the date format. It follows *DateFormat* overwriting rules to overwrite *Javascript configured validators*
DateRange	`data-ft-date-range="1900-01-01:0"`	All "*type="date"*" inputs in the form must match the date range. It follows *DateRange* overwriting rules to overwrite *Javascript configured validators*

3.2 How to validate

To validate your form:

```
if ($('#myForm').formtools('validate')) {
        // form is valid
} else {
        // form is invalid
}
```

3.3 Settings

3.3.1 Javascript configurations

Pay attention to define all javascript configurations before yours js code!

Explicit / Verbose errors

Default value: **true**

With "verbose mode" switched on, error messages will appear after each input tag that contains errors.
You can customize the verbose behaviour in this way:

```
$.fn.formtools.settings.verbose = false;
```

If you want, you can also define a *custom error message container* which is the tag placed after your input to show error.

Error class

Default value: `.has-error`

Since formtools is designed to work with bootsrap, it use by default the bootstrap error class.
If you don't use bootstrap, you can customize the error class in this way:

```
$.fn.formtools.settings.error.class = '.myCustomClass';
```

Parent error class

Default value: `.form-group`

As already said, working on a bootsrap based DOM structure, the default parent class is `.form-group` (parent of input).
If you don't use bootstrap, you can customize the parent error class to achieve your needs.

This class will be searched as the **closest DOM element of the input** and, when found, on input errors it will receives the defined error class.

```
$.fn.formtools.settings.error.parent = '.oneParentInputElement';
```

Custom places for errors

Default value: nothing

If you want to gather errors in specific places, you can set up selectors that will be filled with errors.
If verbose mode is active, errors will be placed **also** under inputs.

```
$.fn.formtools.settings.error.fields = '.oneDiv #myErrorContainer';
```

Global default error message

Default value: nothing

If you want to define a global error message used on all errors of all inputs:

```
$.fn.formtools.settings.error.msg = 'Message for all errors';
```

Define translations

You can define and use custom translations.

To define a translation:

```
$.fn.formtools.settings.translation.en = {
        'ft-required' : 'Required field',
        'ft-minlength' : 'Minimum characters required:',
        'ft-regex' : 'Error',
        'ft-email-validation' : 'Invalid email address',
        'ft-date-validation' : 'Wrong date. Required date format as:',
        'ft-date-range-after' : 'Required date after:',
        'ft-date-range-before' : 'Required date before:'
};

$.fn.formtools.settings.translation.es = { ... };
```

Set up the default language

Default value: **en**

You should set up formtools to use a default language. You can also ask to use a translation defined by you.

```
$.fn.formtools.settings.language = 'en';
```

Validators

Default value: **the only one validator defined by default is** `ftDateFormat="DD/MM/YYYY"`

If you want define default ("global") validators, to apply to all forms:

```
$.fn.formtools.settings.v = {
        'ftDateFormat' : 'MM/DD/YYYY',
        'ftRequired' : ''  // all fields of all forms will be required
};
```

To define any validator in the settings, just take the HTML defined validator name, remove dashes and convert it to camelCase format.
(i.e. data-ft-optional -> ftOptional)

Another one example:

```
$.fn.formtools.settings.v = {
        'ftDateFormat' : 'MM/DD/YYYY',
        'ftRegex' : '([A-Z])\w+'  // all fields of all forms will require at least␣
↪one word containing upper letters
};
```

Hooks

Default value: **no hooks defined**

If you need to do something before formtools actions you can set up hooks in this way:

```
$.fn.formtools.settings.hooks = {
        'preValidate' : function (form) { ... },
        'postValidate' : function (form) { ... },
        'preReset' : function (form, data) { ... },
        'postReset' : function (form, data) { ... }
};
```

Custom error message container

If you need to cutomize the error message placed after the input, you can overwrite this function:

```
$.fn.formtools.formatErrorMsg = function(field, ftErrorId, ftErrorMsg) {
    field.after(
        $('<span>').attr('id', ftErrorId).addClass('help-block ft-error').
↪text(ftErrorMsg)
    );
}
```

Pay attention to give the `ftErrorId` to the element that you generate.
This is really important to allow formtools detach the element from the page when field will become valid.

3.4 Changelog

This file describe new formtools features and incompatibilites

3.4.1 Release 1.0

- Initial release
- Hierarchical configuration
- Custom translations support
- Behaviour customization by rich settings