[**Parsley**](http://docs.google.com/)

[Tweet](https://twitter.com/share)

* [Home](http://docs.google.com/)
* [Examples](http://docs.google.com/examples.html)
* [Documentation](http://docs.google.com/index.html)
* [Download](http://docs.google.com/download.html)
* [Help](http://docs.google.com/help.html)
* [Annotated source](http://docs.google.com/annotated-source/main.html)
* [Tests](http://docs.google.com/tests.html)
* [About](http://docs.google.com/about.html)

Overview

### Frontend form validation

Parsley is a javascript form validation library. It helps you provide your users with feedback on their form submission before sending it to your server. It saves you bandwidth, server load and it saves time for your users.

Javascript form validation is not necessary, and if used, it **does not replace strong backend server validation.**

That's why Parsley is here: to let you define your general form validation, implement it on the backend side, and simply port it frontend-side, with maximum respect to user experience best practices.

Parsley 1.x versions

Parsley's current stable and supported versions are 2.x. If you still use a 1.x version, [here is the related doc](http://parsleyjs.github.io/Parsley-1.x). But don't forget to [upgrade](https://github.com/guillaumepotier/Parsley.js/blob/master/UPGRADE-2.0.md)!

### Data attributes

Parsley uses a specific DOM API which allows you to configure pretty much everything directly from your DOM, without writing a single javascript configuration line or custom function. Parsley's default DOM API is data-parsley-. That means that if in config you see a foo property, it can be set/modified via DOM with data-parsley-foo="value".

### Configuration

You'll see along this documentation and through [examples](http://docs.google.com/examples.html) various available configuration options. You can also view [here](http://docs.google.com/annotated-source/defaults.html) all of Parsley's default configuration options.

# Installation

### Basic installation

Parsley relies on [**jQuery**](http://jquery.com/) **(>= 1.8)**, and it will need to be included before including Parsley.

You will also need to include [**es5-shim**](https://github.com/es-shims/es5-shim) if you want need to support IE8.

Then, you can either use parsley.js unminified file or parsley.min.js minified one. These files and [extras](#2bn6wsx) are [available here](http://docs.google.com/download.html).

Finally, add data-parsley-validate to each <form> you want to be validated.

That would look pretty much like this:

<script src="jquery.js"></script>  
<script src="parsley.min.js"></script>  
  
<form data-parsley-validate>  
...  
</form>

Parsley CSS

Parsley adds many classes and elements in the DOM when it validates. You are strongly encouraged to customize them in your own stylesheets, but here is the "standard" [Parsley css file](http://docs.google.com/src/parsley.css) that is used here on the documentation and examples, if you want to use it to bootstrap your projects with Parsley.

### Javascript installation

Like for [Basic installation](#qsh70q), first include **jQuery** and Parsley. Then, simply use $('#form').parsley(options); or new Parsley('#form', options); (where options is an optional configuration object) to manually bind Parsley to your forms.

That would look pretty much like this:

<script src="jquery.js"></script>  
<script src="parsley.min.js"></script>  
  
<form id="form">  
...  
</form>  
  
<script>  
 $('#form').parsley();  
</script>

Do not add data-parsley-validate to your forms

Please be aware that Parsley looks at all data-parsley-validate occurrences in DOM on document load and automatically binds them if valid.

Once a form or field instance is bound by Parsley, doing $('#form').parsley(options); will update the existing options but not replace them.

### Localization

Parsley comes with various error messages for its built-in validators. They are shipped in English by default, but many other languages are available, thanks to the awesome international Parsley community. [See the available localizations here](https://github.com/guillaumepotier/Parsley.js/tree/master/dist/i18n).

To load a different locale and its messages, include them after Parsley:

<script src="jquery.js"></script>  
<script src="parsley.min.js"></script>  
<script src="i18n/fr.js"></script>  
<script src="i18n/it.js"></script>

The last loaded file will automatically set the messages locale for Parsley. In the example above, we load both French and Italian translations, and use Italian.

### Plugins

Parsley strives to be highly decoupled and modular. It uses events and inheritance, and allows various plugins.

Current available plugins are in [Extras](#2bn6wsx) for additional validators.

# Usage

### Overview

Parsley is a decoupled library that uses different classes to do the heavy work. You'll see here the different protagonists involved and how you can configure them to fit your desired validation.

|  |  |
| --- | --- |
| $ API | Return |
| $('#existingForm').parsley(options) #2.0 | parsleyFormInstance |
| $('#existingInput').parsley(options) #2.0 | parsleyFieldInstance |
| $('#notExistingDOMElement').parsley(options) #2.0 | undefined |
| $('.multipleElements').parsley(options) #2.0 | Array[Instances] |

Look at the source code!

Of course, this documentation tries to be the most exhaustive possible and relatively easy to understand. This documentation also provides the [complete annotated source](http://docs.google.com/annotated-source/main.html). Please take 5 minutes of your time to have a quick glance at it, and at least understand the architecture (Parsley, ParsleyForm, ParsleyField, ParsleyValidator, ParsleyUI, Utils, Pub/Sub..), it will heavily ease the lecture below.

### Configuration

#### Data attributes and javascript

The multiple options can be specified using data attributes and javascript:

<input id="first" data-parsley-maxlength="42" value="hello"/>  
<input id="second" value="world"/>  
[...]  
<script>  
var instance = $('#first').parsley();  
console.log(instance.isValid()); // maxlength is 42, so field is valid  
$('#first').attr('data-parsley-maxlength', 4);  
console.log(instance.isValid()); // No longer valid, as maxlength is 4  
// You can access and override options in javascript too:  
instance.options.maxlength++;  
console.log(instance.isValid()); // Back to being valid, as maxlength is 5  
// Alternatively, the options can be specified as:  
var otherInstance = $('#second').parsley({  
 maxlength: 10  
});  
console.log(otherInstance.options); // Shows that maxlength will be 10 for this field

As shown in the previous example, Parsley will actualize the options from the data attributes whenever it needs to validate a field.

#### Option inheritance

Field instances inherit their options from Form instances, and both inherit from the global options. This is a handy way to configure all your form's inputs in a row by passing their config through form.

<form>  
 <input/>  
</form>  
[...]  
<script>  
Parsley.options.maxlength = 42;  
var formInstance = $('form').parsley();  
var field = $('input').parsley();  
console.log(field.options.maxlength); // Shows that maxlength is 42  
Parsley.options.maxlength = 30;  
console.log(field.options.maxlength); // Shows that maxlength is automagically 30  
formInstance.options.maxlength++;  
console.log(field.options.maxlength); // Shows that maxlength is automagically 31

The previous example shows that the inheritance of options is automagic. In case you are wondering, they are linked through prototype to achieve this.

#### Naming

You can change the DOM API namespace directly using the namespace option. Data attributes with compound names are camelcased for javascript, and their values are automatically converted to the appropriate type (boolean, integer, etc.). For example:

<input data-my-namespace-priority-enabled="false">  
[...]  
<script>  
var instance = $('input').parsley({namespace: 'my-namespace-'});  
if (false === instance.options.priorityEnabled)  
 console.log("priorityEnabled was set to false");

### Form

When doing $('#target').parsley() or new Parsley('#target'); on a <form id="target"> element, it will bind the whole form and its various inputs and return a ParsleyForm instance.

#### Options

|  |  |  |
| --- | --- | --- |
| Property | Default | Description |
| data-parsley-namespace #2.0 | data-parsley- | Namespace used by Parsley DOM API to bind options from DOM.  [See more](#3as4poj) |
| data-parsley-validate #2.0 |  | Auto bind your form with Parsley validation on document load. |
| data-parsley-priority-enabled #2.0 | true | Either validate higher priority constraints first and stop on first failure (true), or validate all constraints simultaneously and show all the failing ones (false). |
| data-parsley-inputs #2.0 | input,  textarea,  select | When looking for fields within a form, Parsley uses this selector. The fields found will then be filtered using the excluded option below. |
| data-parsley-excluded #2.0 | input[type=button],  input[type=submit],  input[type=reset],  input[type=hidden] | Form fields that won't be validated by Parsley. For example, if you want to add disabled and hidden fields to the existing list, use:  data-parsley-excluded="input[type=button], input[type=submit], input[type=reset], input[type=hidden], [disabled], :hidden" |

#### Methods

|  |  |  |
| --- | --- | --- |
| Method | Returns | Description |
| whenValid({group, force}) #2.2 | promise | Returns a jQuery promise that will be fulfilled if and only if the Form is valid. **Does not affect UI nor fires** [**events**](#1pxezwc)**.** If group is given, it only validates fields that belong to this group. If force is given, it force validates even non-required fields ([See example](http://docs.google.com/examples/events.html)) |
| isValid({group, force}) #2.0 | boolean or null | Similar to whenValid but returns true if the promise is already fulfilled, false if already rejected, and null if the validation is still pending. |
| whenValidate({group, force}) #2.0 | promise | Validate form. Prevents submission if not valid. **Fires** [**events**](#1pxezwc) **and affects UI.**. You can only validate a certain group of fields by specifying optional group string parameter. If group is given, it only validates fields that belong to this group. If force is given, it force validates even non required fields ([See example](http://docs.google.com/examples/events.html)). Same result as whenValid. |
| validate({group, force}) #2.0 | boolean or null | Same as whenValidate except it returns true if the promise is already fulfilled, false if already rejected, and null if the validation is still pending. |
| refresh() #2.8 |  | Forces a refresh of the form and its field. Parsley always refreshes before validation, but this may be helpful for dynamic changes that need to be applied immediately (e.g. dynamically adding an input with a trigger, changing the `inputs` or `excluded` options, etc.). |
| reset() #2.0 |  | Reset UI for this form and for its fields. |
| destroy() #2.0 |  | Disable and destroy Parsley for this form and its fields. |

#### UI

### See [UI for Form](#49x2ik5) section.

Field

When doing $('#target').parsley() or new Parsley('#target'); on a <input id="target"> element (or <textarea>, <select>), it will bind the field and return a ParsleyField instance. Except for input types radio and checkbox that don't have a name attribute or a data-parsley-multiple attribute, they won't be bound (ignored) and will eventually raise a warning in the console.

#### Options

|  |  |
| --- | --- |
| Property | Description |
| data-parsley-value #2.0 | Set a specific field value for Parsley validation, dissociated from the real one. eg: data-parsley-value="foo"  The JavaScript API allows one to pass a function to be called. eg: $('<input type="text">').appendTo($('form')).  parsley({    value: function(parsley) { return 'foo'; }  }); |
| data-parsley-group #2.0 | Assign a group to a field for specific group validation. eg: data-parsley-group="signup". This way, you could only validate a portion of a form and not all the fields. Can be multiple. eg: data-parsley-group='["foo", "bar"]' |
| data-parsley-multiple #2.0 | You can add this attribute to radio / checkboxes elements like this: data-parsley-multiple="mymultiplelink" to link them together even if they don't share the same name. |
| data-parsley-validate-if-empty #2.0 | A field is by default not validated if it is not required and empty. By adding data-parsley-validate-if-empty, validation will be done even if field is empty. Useful if you need some custom validators that check something particular when a field is empty. |
| data-parsley-whitespace #2.1 | Perform actions on whitespace in value **only for Parsley validation** (and not inside the input itself, data sent by your form won't be edited). Useful if your backend already does so and if extra whitespace could unnecessarily mess with your validation.  Use: data-parsley-whitespace="trim" to trim leading and trailing whitespace characters.  Use: data-parsley-whitespace="squish" to squish multiple sequential whitespace characters into a single whitespace character, and also trim leading and trailing whitespace characters. |
| data-parsley-ui-enabled #2.0 | If set to false, Parsley will not affect UI for this field. |
| data-parsley-errors-messages-disabled #2.0 | Add parsley-success and parsley-error classes on field, but no error message. |
| data-parsley-excluded #2.1 | If set to true, Parsley will ignore this field when binding a form. |
| data-parsley-debounce #2.3 | Postpones validation for a given number of milliseconds after user input has stopped arriving, eg: data-parsley-debounce="500". Useful for expensive validations (such as remotes) that you don't want to run on every keystroke. |

Checkbox, radio and select multiple

These fields are a bit different than regular input, textarea or simple select. They need to have either a name or an id attribute to be correctly bound and validated by Parsley. Otherwise, they will be ignored and a warning will be put in the console.

#### Methods

|  |  |  |
| --- | --- | --- |
| Method | Returns | Description |
| isValid({force}) #2.0 | true if all ok  null if some validations are still pending  [Violation, ...]] if fails | Returns if the field is valid or not. **Does not affect UI nor fires** [**events**](#1pxezwc)**.** If force is set, it forces validation even on non-required fields ([See example](http://docs.google.com/examples/events.html)) |
| validate({force, group}) #2.0 | true if all ok  null if some validations are still pending  [Violation, ...]] if fails | Validate Field. **Fires** [**events**](#1pxezwc) **and affects UI.** If force is set, force validate even non required fields ([See example](http://docs.google.com/examples/events.html)) |
| getErrorsMessages() #2.0 | array | Returns an array of field's error messages |
| reset() #2.0 |  | Reset UI for this field. |
| destroy() #2.0 |  | Disable and destroy Parsley for this field. |

#### UI

# See [UI for Field](#2p2csry) section.

Built-in validators

### Overview

A *validator* is a method to determine if a given *value* (or sometimes sets of values) is acceptable or not, given some *requirement* parameters.

Parsley comes with many builtin validators and provides tools to specify your own.

### Builtin validators list

|  |  |  |
| --- | --- | --- |
| Name | API | Description |
| Required #2.0 | |  |  | | --- | --- | | required | HTML5 | | data-parsley-required |  | | data-parsley-required="true" |  | | data-parsley-required="false" |  | | Validates that a required field has been filled with a non blank value. If data-parsley-required="false", validator is deactivated and the field is not required. |
| Email #2.0 | |  |  | | --- | --- | | type="email" | HTML5 | | data-parsley-type="email" |  | | Validates that a value is a valid email address. |
| Number #2.0 | data-parsley-type="number" | Validates that a value is a valid number according to the given step, min and original value.  The default step for HTML5 is "1", which is so counterintuive that Parsley uses a default step of "any" for data-parsley-type="number". Warning! HTML5 type="number" can be counterintuitive. The default step of '1' is near useless. Moreover, for browsers that support it, the value accessible from javascript in case of invalid input is "", so you will never get an error (unless it is also required). |
| Integer #2.0 | |  |  | | --- | --- | | type="number" | HTML5 | | data-parsley-type="integer" |  | | Validates that a value is a valid integer. |
| Digits #2.0 | data-parsley-type="digits" | Validates that a value is only digits. |
| Alphanum #2.0 | data-parsley-type="alphanum" | Validates that a value is a valid alphanumeric string. |
| Url #2.0 | |  |  | | --- | --- | | type="url" | HTML5 | | data-parsley-type="url" |  | | Validates that a value is a valid url. |
| Minlength #2.0 | |  |  | | --- | --- | | minlength="6" | HTML5 | | data-parsley-minlength="6" |  | | Validates that the length of a string is at least as long as the given limit. |
| Maxlength #2.0 | |  |  | | --- | --- | | maxlength="6" | HTML5 | | data-parsley-maxlength="6" |  | | Validates that the length of a string is not longer than the given limit. |
| Length #2.0 | data-parsley-length="[6, 10]" | Validates that a given string length is between some minimum and maximum value. |
| Min #2.0 | |  |  | | --- | --- | | min="6" | HTML5 | | data-parsley-min="6" |  | | Validates that a given input (number or date) or date is greater than or equal to some minimum (number or date.) |
| Max #2.0 | |  |  | | --- | --- | | max="10" | HTML5 | | data-parsley-max="6" |  | | Validates that the given input (number or date) is less than or equal to some maximum value (number or date). |
| Range #2.0 | |  |  | | --- | --- | | type="range" | HTML5 | | data-parsley-range="[6, 10]" |  | | Validates that a given value (number or date) is between some minimum and maximum values (numbers or dates). |
| Pattern #2.0 | |  |  | | --- | --- | | pattern="\d+" | HTML5 | | data-parsley-pattern="\d+" |  | | Validates that a value matches a specific regular expression (regex).  Note that patterns are anchored, i.e. must match the whole string.  Parsley deviates from the standard for patterns looking like /*pattern*/*{flag}*; these are interpreted as literal regexp and are not anchored. |
| MinCheck #2.0 | data-parsley-mincheck="3" | Validates that a certain minimum number of checkboxes in a group are checked. |
| MaxCheck #2.0 | data-parsley-maxcheck="3" | Validates that a certain maximum number of checkboxes in a group are checked. |
| Check #2.0 | data-parsley-check="[1, 3]" | Validates that the number of checked checkboxes in a group is within a certain range. |
| Equalto #2.0 | data-parsley-equalto="#anotherfield" | Validates that the value is identical to another field's value (useful for password confirmation check). |

These validators are shipped in parsley.js. Have a look at [Parsley Extras](#2bn6wsx) for more validators.

# Custom Validators

### Craft yours

Of course, Parsley built-in validators are commonly used validators, and you'll need some more that fit your specific forms and validations. That's why Parsley lets you easily create your own validators.

The preferred way to register them (after parsley.js is loaded) looks like:

<input type="text" data-parsley-multiple-of="3" />  
[...]  
<script>  
window.Parsley  
 .addValidator('multipleOf', {  
 requirementType: 'integer',  
 validateNumber: function(value, requirement) {  
 return 0 === value % requirement;  
 },  
 messages: {  
 en: 'This value should be a multiple of %s',  
 fr: 'Cette valeur doit être un multiple de %s'  
 }  
 });  
</script>

The following sections go over the details on how to define a custom validator

### Validating function

There are many ways a validator can specify how to validate data:

|  |  |
| --- | --- |
| Name | Description |
| validateString | Is passed the input's value as a string. |
| validateNumber | Use this instead of validateString when only numeric values are acceptable. Parsley will parse the input's value and pass the number, or reject the value if it's not an acceptable number |
| validateDate | Define this to treate date values. Parsley will parse the input's value and pass the date, or reject the value if it's not an acceptable date. The format used must be that [of the standard](https://html.spec.whatwg.org/multipage/infrastructure.html#valid-date-string), e.g. "2017-02-28". |
| validateMultiple | Is passed an array of values, in the case of checkboxes. |

Your custom validator must specify at least one of these. If it can validate both single inputs and multiple (i.e. checkboxes), then you can specify *validateMultiple* and one of the other two.

Validating functions should return either true if the value is valid, or false otherwise. It can instead return a jQuery promise that will be resolved if the value is valid, or be rejected otherwise.

Validators can reject with a custom error message as a first argument if desired.

### Requirement parameters

You can specify what kind of requirement parameter your custom validator is expecting:

|  |  |
| --- | --- |
| Value of requirementType | Description |
| 'string' | The most generic kind; requirement passed as is, with no checking. |
| 'integer' | For integers only (e.g. used by minlength) |
| 'number' | To be used when decimal numbers are acceptable |
| 'date' | To be used for dates. The format used must be that [of the standard](https://html.spec.whatwg.org/multipage/infrastructure.html#valid-date-string), e.g. "2017-02-28". |
| 'regexp' | Requirement can be either a full regexp string (e.g. /hel+o/i) or just a simple expression (e.g. hel+o) |
| 'boolean' | Any value other than "false" will be considered to true, including the empty string. This is so data-parsley-required and data-parsley-required=true be treated the same way. |

You can also specify an array of these kinds. For example, if a validator has requirementType: ['integer', 'integer'], then given the requirement string "[1, 2]" it will receive 1 and 2 as second and third arguments (the first one being the value(s) to validate).

For cases where more complex parameters are needed, you can specify extra parameters; refer to the source and check how the remote validator uses that.

### Error messages

You can specify error messages, in as many locales as desired, using the messages option.

This is equivalent to calling addMessage for each locale.

# UI/UX

### Overview

Parsley ships a UI/UX component that is the only one responsible for classes, error messages, focus or trigger events that alter your page. It strives to be the most UX friendly. Here are the main mottos for ParsleyUI:

1. **Min char validation**: Parsley by default does not proceed with field validation when less than 3 chars have been input. Do not assault your users with error messages too soon!
2. **One error at the time**: constraints are prioritized in Parsley, and if several of them are not met for a field on validation, only show the most important one.
3. **Quick error removal**: when a field is detected and shown as invalid, further checks are done on each keypress to try to quickly remove error messages once the field is ok.
4. **Control focusing on error**: on form submission, the first error field is focused to allow the user to easily start fixing errors.

Naturally, all of this is absolutely customizable; you'll see below how to configure your desired UX behavior.

### Classes and templates

Parsley adds its share of classes and elements, to ease nice UI validation result display. By default, it will add parsley-success and parsley-error classes depending on the validation result, **on the input itself for a simple text, textarea and select input, and on the parent for radio / checkboxes inputs.**

Customize your classes

You could change these classes' names in configuration, and the class holder element too.

### UI for form

|  |  |  |
| --- | --- | --- |
| Name | API | Description |
| UI Enabled #2.0 | data-parsley-ui-enabled="false" | Activate or deactivate UI |
| Focus #2.0 | data-parsley-focus="first" | Focus failing field on form validation. Possible values: 'first' | 'last' | 'none' |

### UI for field

|  |  |  |
| --- | --- | --- |
| Name | API | Description |
| Trigger #2.0 | data-parsley-trigger="input" | Specify one or many javascript events that will trigger item validation, before any failure. To set multiple events, separate them with a space data-parsley-trigger="focusin focusout". Default is null. [See the various events supported by jQuery.](http://api.jquery.com/category/events/) |
| Trigger After Failure #2.0 | data-parsley-trigger-after-failure="focusout" | Specify one or many javascript events that will trigger item validation, after the first failure. Default is ['input'](https://developer.mozilla.org/en-US/docs/Web/Events/input). |
| No focus #2.0 | data-parsley-no-focus | If this field fails, do not focus on it (if first focus strategy is on, next field would be focused, if last strategy is on, previous field would be focused) |
| Validation threshold #2.0 | data-parsley-validation-threshold="10" | Used with trigger option above, for all key- events, do not validate until the field has a certain number of characters. Default is 3 |
| Class handler #2.0 | data-parsley-class-handler="#element" | Specify the existing DOM container where ParsleyUI should add error and success classes. It is also possible to configure it with a callback function from javascript, see [the annotated source](http://docs.google.com/annotated-source/defaults.html). |
| Errors container #2.0 | data-parsley-errors-container="#element" | Specify the existing DOM container where ParsleyUI should put the errors. It is also possible to configure it with a callback function from javascript, see [the annotated source](http://docs.google.com/annotated-source/defaults.html). |
| Error message #2.0 | data-parsley-error-message="my message" | Customize a unique global message for the field. |
| Validator error message #2.0 | data-parsley-`constraint`-message="my message" | Customize the error message for the field constraint. eg: data-parsley-required-message="this field is required" |

### UI for javascript

Field instances have low level methods to add, update & remove manage error messages. You have to manage these errors completely manually and they should be independant with other validations. Prefer using custom validations whenever possible. Note that getErrorsMessages only return errors from validations and not manually added errors.

|  |  |  |
| --- | --- | --- |
| Name | Method | Description |
| Add error #2.0 | addError(name, {message: , assert: , updateClass: true}); | Add an error message. |
| Update error #2.0 | updateError(name, {message: , assert: , updateClass: true}); | Update an already added error message. |
| Remove error #2.0 | removeError(name, {updateClass: true}); | Remove an already present error message. |

# Events

### Overview

Parsley triggers events that allows ParsleyUI to work. Further more, it could allow you to do some powerful magic if you listen properly to the right events!

For performance reasons, Parsley does not use jQuery events, but the API to listen to events is very similar:

$('#some-input').parsley().on('field:success', function() {  
 // In here, `this` is the parlsey instance of #some-input  
});

Similarly to jQuery events, parsley events will bubble up. For example, if a field is about to be validated, the event 'field:validate' will be triggerred first on the parsley field instance, then on the parsley form instance (if the field is bound in a form) and finally on the top level window.Parsley

window.Parsley.on('field:error', function() {  
 // This global callback will be called for any field that fails validation.  
 console.log('Validation failed for: ', this.$element);  
});

### .

Events List

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Instance | Fired by | Description |
| form:init #2.1 | ParsleyForm | new Parsley() | Triggered when a Form is bound for the first time. |
| form:validate #2.1 | ParsleyForm | .validate() | Triggered when a form validation is triggered, **before** its validation. |
| form:success #2.1 | ParsleyForm | .validate() | Triggered when a form validation is triggered, **after** its validation succeeds. |
| form:error #2.1 | ParsleyForm | .validate() | Triggered when a form validation is triggered, **after** its validation fails. |
| form:validated #2.1 | ParsleyForm | .validate() | Triggered when a form validation is triggered, **after** its validation finishes (with success or with errors). |
| form:submit #2.2 | ParsleyForm | submit() | Triggered when after a form validation succeeds and before the form is actually submitted.  Return `false` to interrupt submission. |
| field:init #2.1 | ParsleyField | new Parsley() | Triggered when a Field is bound for the first time. |
| field:validate #2.1 | ParsleyField | .validate() | Triggered when a field validation is triggered, **before** its validation. |
| field:success #2.1 | ParsleyField | .validate() | Triggered when a field validation succeeds. |
| field:error #2.1 | ParsleyField | .validate() | Triggered when a field validation fails. |
| field:validated #2.1 | ParsleyField | .validate() | Triggered after a field is validated (with success or with errors). |

# Parsley Remote

Parsley [remote](http://docs.google.com/annotated-source/remote.html) is an easy to use **ajax asynchronous validator**.

### Options

|  |  |  |
| --- | --- | --- |
| Name | API | Description |
| Remote validator | data-parsley-remote #2.0 | Define the url that will be called to validate the entered content. e.g. data-parsley-remote="http://url.ext".  If the url contains the string "{value}", the value will replace it in the URL (typical of RESTful APIs), otherwise the value will be passed as a data parameter, with the key being the input's name or ID. |
| Reverse | data-parsley-remote-reverse #2.0 | By default, all 2xx ajax returs are considered valid, all others failure. Sometimes (when a call is needed to see if an email, a pseudo is available for example) a 404 API answer could be the right answer. Using data-parsley-remote-reverse="true" will consider 200 response is an error, and 404 one is correct. |
| Options | data-parsley-remote-options #2.0 | You could pass a json object to the $.ajax() method used by remote validator. eg:  data-parsley-remote-options='{ "type": "POST", "dataType": "jsonp", "data": { "token": "value" } }'  Warning: you must format your JSON string wrapping all the keys/values with " like above otherwise it won't be correctly parsed by $.parseJSON() used behind the scenes by remote validator ([See jQuery doc](https://api.jquery.com/jQuery.parseJSON/)) |
| Validator | data-parsley-remote-validator #2.0 | Use a specific remote validator. By default, there are 2 built-in remote validators: default and reverse. Default one is used by default and Reverse one used when data-parsley-remote-reverse is set to true. (this is an alias, you could use data-parsley-remote-validator="reverse").  Inside the function, this keyword refers to the ParsleyField instance attached to the form element. You have access to the plugin as well as the element if you need to perform other actions before returning the validation result.  To learn how to craft your custom remote validators, go [here](#147n2zr). |

### Events

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Instance | Fired by | Description |
| field:ajaxoptions #2.2 | ParsleyField | whenIsValid & al. | Triggered just before an ajax request is sent, so one can tweak the options passed to $.ajax. Options are passed as a second parameter. |

### Methods

|  |  |
| --- | --- |
| Method | Description |
| Parsley.addAsyncValidator(name, fn) #2.0 | Specify custom validator for Ajax results. |

### Custom remote validators

If you need some custom processing of Ajax responses, configure your custom remote as follows:

<input name="q" type="text" data-parsley-remote data-parsley-remote-validator='mycustom' value="foo" />  
[...]  
<script href="parsley.remote.js"></script>  
<script>  
window.Parsley.addAsyncValidator('mycustom', function (xhr) {  
 console.log(this.$element); // jQuery Object[ input[name="q"] ]  
  
 return 404 === xhr.status;  
 }, 'http://mycustomapiurl.ext');  
</script>

### Combining Remote Validations with Groups

If you need to trigger validate outside of form submission, such as with data-parsley-group="group-name", you'll need to make use of the promises provided in whenValidate({group, force}). The validate({group, force}) method that returns a boolean or null will always return null due to remote validation always returning open promises.

<script type="text/javascript">  
$("form").parsley().whenValidate({  
 group: 2  
}).done(function() {  
 // trigger step change  
});  
</script>

# Parsley Extras

You'll find in the src/extra/ directory in Parsley .zip or Github projects many more or less useful validators crafted by the community. A doc here is coming.

### Validators list

|  |  |  |
| --- | --- | --- |
| Name | API | Description |
| Greater than #2.0 | |  |  | | --- | --- | | data-parsley-gt="#anotherfield" |  | | data-parsley-gt="6" |  | | Validates that the value is greater than another field's value or some strict minimum number. |
| Greater than or equal to #2.0 | |  |  | | --- | --- | | data-parsley-gte="#anotherfield" |  | | data-parsley-gte="6" |  | | Validates that the value is greater than or equal to another field's value or some minimum number. |
| Less than #2.0 | |  |  | | --- | --- | | data-parsley-lt="#anotherfield" |  | | data-parsley-lt="6" |  | | Validates that the value is less than another field's value or some strict maximum number. |
| Less than or equal to #2.0 | |  |  | | --- | --- | | data-parsley-lte="#anotherfield" |  | | data-parsley-lte="6" |  | | Validates that the value is less than or equal to another field's value or some maximum number. |
| Minwords #2.0 | |  |  | | --- | --- | | data-parsley-minwords="200" |  | | Validates that the value have at least a certain amount of words |
| Maxwords #2.0 | |  |  | | --- | --- | | data-parsley-maxwords="200" |  | | Validates that the value have a maximum of a certain amount of words |
| Words #2.0 | |  |  | | --- | --- | | data-parsley-words="[200, 600]" |  | | Validates that the value is within a certain range of words |

* [Overview](#3o7alnk)
  + [Frontend form validation](#23ckvvd)
  + [Data attributes](#3as4poj)
  + [Configuration](#ihv636)
* [Installation](#32hioqz)
  + [Basic installation](#qsh70q)
  + [Javascript installation](#1hmsyys)
  + [Localization](#41mghml)
  + [Plugins](#2grqrue)
* [Usage](#vx1227)
  + [Overview](#3fwokq0)
  + [Configuration](#1v1yuxt)
  + [Form](#4f1mdlm)
  + [Field](#2u6wntf)
* [Built-in validators](#19c6y18)
  + [Overview](#3tbugp1)
  + [Built-in validators list](#28h4qwu)
* [Custom validators](#nmf14n)
  + [Craft yours!](#37m2jsg)
  + [Validating functions](#1mrcu09)
  + [Requirement parameters](#46r0co2)
  + [Error messages](#2lwamvv)
* [UI/UX](#111kx3o)
  + [Overview](#3l18frh)
  + [Class & templates](#206ipza)
  + [UI for Form](#49x2ik5)
  + [UI for Field](#2p2csry)
  + [UI for javascript](#4k668n3)
* [Events](#1pxezwc)
  + [Overview](#2zbgiuw)
  + [List](#1egqt2p)
* [Parsley Remote](#3ygebqi)
  + [Options](#2dlolyb)
  + [Events](#sqyw64)
  + [Methods](#3cqmetx)
  + [Custom remote validators](#147n2zr)
* [Parsley Extras](#2bn6wsx)
  + [Validators list](#1rvwp1q)

[Back to top](#4bvk7pj)

Subscribe to Parsley mailing list Receive a mail each 2.x new release, and sometimes an email asking for your feedback (twice a year max!)

© [Guillaume Potier](https://twitter.com/guillaumepotier) 2014 - [@Wisembly](http://wisembly.com)