Parsley (../)

Overview

Frontend form validation

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

Javascript form validation is not necessary, and if used, it **does not replace a 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 could be set/modified via DOM with `data-parsley-foo="value"` .

> ### More on Parsley DOM API
>
> 1. For composed property names, Parsley uses camelization, like jQuery `$.data()` does. That means that `data-parsley-composed-property` will correspond in code to `composedProperty`, and vice versa.
> 2. Every property set in DOM will be stored in `this.options` for each Parsley instance.
> 3. You can change the DOM API namespace directly in DOM (inception), by doing `data-parsley-namespace="data-my-namespace-"` and then your Parsley DOM attributes will look like `data-my-namespace-property="value"`.
> 4. Special chars: if you need to reference names with special characters, like `id="this:is:a:special:id"`, you'll need to use this syntax in the DOM API: `data-parsley-errors-container='div[id="this:is:a:special:id"]'` otherwise it won't work.

Configuration

You'll see along this documentation and through examples (examples.html) various available configuration options. You can also view here (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 would need to be included before including Parsley.

Then, you can either use `parsley.js` unminified file or `parsley.min.js` minified one. These files and other builds (Remote, Extras ..) are available here (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 (../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, 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 type="text/javascript">
  $('#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, default instantiation options cannot be easily updated, only the ones defined in the DOM would work. Which means that if this DOM validation attribute is present, doing `$('#form').parsley(options);` would just return the automatically bound instance, and not the one you would expect with the `options` you wanted to set.

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/src/i18n).

To load a different locale and its messages, you have two possibilities:

- Load your needed localization **before** parsley, then select the one you need once Parsley is loaded and if you want to switch from the English default. In this example, we'll load both French and Italian translations, and use French:

```
<script src="jquery.js"></script>
<script src="i18n/fr.js"></script>
<script src="i18n/it.js"></script>
<script src="parsley.min.js"></script>
<script type="text/javascript">
  window.ParsleyValidator.setLocale('fr');
</script>
```

- Load your needed localization **after** parsley. The last loaded file will automatically set the messages locale for ParsleyValidator. In this example, we'll load both French and Italian translations, and use Italian:

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

Plugins

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

Current available plugins are:

- Parsley Remote: provides an ajax validator and asynchronous validation
- Parsley Extras: provides some extras and useful 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 (annotated-source/parsley.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.

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 you a `ParsleyForm` instance.

> ### Field options inheritance

> All the options you pass through DOM or constructor for a form will be inherited by every `ParsleyField` input instance that belongs to the form. This is a handy way to configure all your form's inputs in a row by passing their config through form.

Options

Property	Default	Description
`data-parsley-namespace` #2.0	`data-parsley-`	Namespace used by Parsley DOM API to bind options from DOM. See more
`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-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
`isValid(group, force)` #2.0	`boolean`	Returns if the Form is valid or not. **Does not affect UI nor fires events.** 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 (examples/events.html))
`validate(group, force)` #2.0	`boolean`	Validate form. Prevents submission if not valid. **Fires events 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 (examples/events.html))
`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 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 you a `ParsleyField` instance. Except for input types radio and checkbox that does not have a `name` attribute or a `data-parsley-multiple` attribute, they won't be bound (ignored) and would 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"`
`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-trim-value` #2.0	Trim field value **only for Parsley validation** (and not inside the input itself, data sent by your form won't be trimmed). Useful if your backend already does so and if trailing spaces could unnecessarily mess with your validation. Use: `data-parsley-trim-value="true"`.
`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.

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 would be ignored and a warning will be put in the console.

Methods

Method	Returns	Description
`isValid(force)` #2.0	`true` if all ok `[]` if empty optional field `[Violation [, Violation..]]`	Returns if the Field is valid or not. **Does not affect UI nor fires events.** If `force` is given, it forces validation even on non required fields (See example (examples/events.html))

	if fails	
`validate(force)` #2.0	`true` if all ok `[]` if empty optional field `[Violation [, Violation..]]` if fails	Validate Field. **Fires events and affects UI.** If `force` is given, force validate even non required fields (See example (examples/events.html))
`reset()` #2.0		Reset UI for this field.
`destroy()` #2.0		Disable and destroy Parsley for this field.

UI

See UI for Field section.

Global configuration

It is possible to configure Parsley options globally and avoid passing them to each form on your site. To do so, you'll need to define a `window.ParsleyConfig = {}` object **before** calling Parsley file.

For example, if you want to customize the field errors and use div and spans instead of ul and lis, use this snippet:

```
<script>
window.ParsleyConfig = {
  errorsWrapper: '<div></div>',
  errorTemplate: '<span></span>'
};
</script>
<script src="parsley.js"></script>
```

Built-in validators

Overview

Parsley 2.x is now based on validator.js (http://validatorjs.org) that ships commonly used validators and simplifies group and priority validation. That way, Parsley mutualize validators with validators.js and define new ones using the `Callback()` Assert.

> ### Use Validator.js in your Parsley projects
>
> Parsley let you use validator.js in `window.ParsleyValidator.Validator`

Validators list

Name	API	Description
Required	`required` **HTML5**	Validate that a required field has been filled with a non blank

#2.0	`data-parsley-required`
	`data-parsley-required="true"`
	`data-parsley-required="false"`
		value. If `data-parsley-required="false"` , validator is deactivated and field 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. **Warning!** HTML5 `type="number"` is binded with below `integer` validator.
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 larger 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 number is greater than or equal to some minimum number.
Max #2.0	`max="10"` **HTML5** `data-parsley-max="6"`	Validates that a given number is less than or equal to some maximum number.
Range #2.0	`type="range"` **HTML5** `data-parsley-range="[6,10]"`	Validates that a given number is between some minimum and maximum number.
Pattern #2.0	`pattern="\d+"` **HTML5** `data-parsley-pattern="\d+"`	Validates that a value matches a specific regular expression (regex).
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 Remote plugin and Parsley Extras for more 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.

Here again, like localizations, configuring your custom validators and error messages comes with two flavors:

- By registering them in some globals **before** calling `parsley.js`:

```
<input type="text" data-parsley-multiple="3" />
[...]
<script type="text/javascript">
window.ParsleyConfig = {
  validators: {
    multiple: {
      fn: function (value, requirement) {
        return 0 === value % requirement;
      },
      priority: 32
    }
  },
  i18n: {
    en: {
      multiple: 'This value should be a multiple of %s'
    },
    fr: {
      multiple: 'Cette valeur doit être un multiple de %s'
    }
  }
};
</script>
```

- By registering them in `ParsleyValidator` **after** `parsley.js` is loaded:

```
<input type="text" data-parsley-multiple="3" />
[...]
<script type="text/javascript">
window.ParsleyValidator
  .addValidator('multiple', function (value, requirement) {
    return 0 === value % requirement;
  }, 32)
  .addMessage('en', 'multiple', 'This value should be a multiple of %s')
  .addMessage('fr', 'multiple', 'Cette valeur doit être un multiple de %s');
</script>
```

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 many 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 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="change"`	Specify one or many javascript events that will trigger item validation. To set multiple events, separate them by a space `data-parsley-trigger="focusin focusout"` . See the various events supported by jQuery. (http://api.jquery.com/category/events/)

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 field have 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 (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 (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

Name	Method	Description
Add error #2.0	`window.ParsleyUI.addError(parsleyInstance, name, message);`	Manually add an error message.
Update error #2.0	`window.ParsleyUI.updateError(parsleyInstance, name, message);`	Manually edit an error message.
Remove error #2.0	`window.ParsleyUI.removeError(parsleyInstance, name);`	Manually remove an error message.
Get errors messages #2.0	`window.ParsleyUI.getErrorsMessages(parsleyInstance);`	Returns an array of the field errors messages displayed once validated.

Events

Overview

Parsley comes with a tiny pub/sub mechanism (annotated-source/pubsub.html) that allows ParsleyUI to work. Further more, it could allow you to do some powerful magic if you listen properly to the right events!

Events List

Name	Instance	Fired by	Description
`parsley:form:init` #2.0	`ParsleyForm`	`new Parsley()`	Fired when a Form is bound for the first time.
`parsley:form:validate` #2.0	`ParsleyForm`	`.validate()`	Fired when a form validation is triggered, **before** its validation.
`parsley:form:success` #2.0	`ParsleyForm`	`.validate()`	Fired when a form validation is triggered, **after** its validation succeeds.
`parsley:form:error` #2.0	`ParsleyForm`	`.validate()`	Fired when a form validation is triggered, **after** its validation fails.
`parsley:form:validated` #2.0	`ParsleyForm`	`.validate()`	Fired when a form validation is triggered, **after** its validation finishes (with success or with errors).
`parsley:field:init` #2.0	`ParsleyField`	`new Parsley()`	Fired when a Field is bound for the first time.
`parsley:field:validate` #2.0	`ParsleyField`	`.validate()`	Fired when a field validation is triggered, **before** its validation.
`parsley:field:success` #2.0	`ParsleyField`	`.validate()`	Fired when a field validation succeeds.
`parsley:field:error` #2.0	`ParsleyField`	`.validate()`	Fired when a field validation fails.
`parsley:field:validated` #2.0	`ParsleyField`	`.validate()`	Fired after a field is validated (with success or with errors).

Usage

Method	Description
`$.listen('name', callback)` #2.0	Listen to a specific event. Fire a callback function.
`$.listen('name', context, callback)` #2.0	Listen to a specific event. Fire a callback function that would have a specific context.
`.subscribe('name', callback)` #2.0	On a `ParsleyField`, it would only be fired if the specific event is fired by the specific field. On a `ParsleyForm`, it would be fired if the event is fired by the form, or **by any of its fields**.

`.unsubscribe('name')` #2.0	Unsubscribe all callbacks subscribed to an event name.
`$.emit('name')` #2.0	Emit an event for listeners and subscribers.

Parsley Remote

Parsley remote (annotated-source/parsley.remote.html) is a handy plugin that adds a **unique ajax asynchronous validator**.

To use this plugin, either load `parsley.remote.js` **before** loading `parsley.js`, or directly load `parsley.remote.js`.

Options

Name	API	Description
Remote validator	`data-parsley-remote` #2.0	Define the url that would be called to validate the entered content. eg: `data-parsley-remote="http://url.ext"`
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.

Methods

Method	Description
`asyncIsValid()` #2.0	Asynchronously get if field or form is valid or not. Returns a jQuery promise.
`asyncValidate()` #2.0	Asynchronously validate a field or a form and display UI errors. Returns a jQuery promise.
`addAsyncValidator(name, fn)` #2.0	Add a new custom remote validator.

Custom remote validators

Configuring your custom remote validators comes with two solutions:

- By registering them in some globals **before** calling `parsley.remote.js` :

```
<input name="q" type="text" data-parsley-remote data-parsley-remote-validator='mycustom' value="foo"
 />
[...]
<script type="text/javascript">
window.ParsleyExtend = {
  asyncValidators: {
    mycustom: {
      fn: function (xhr) {
        console.log(this.$element); // jQuery Object[ input[name="q"] ]

        return 404 === xhr.status;
      },
      url: 'http://mycustomapiurl.ext'
    }
  }
};
</script>
<script href="parsley.remote.js"></script>
<script href="parsley.js"></script>
```

- By registering them **after** `parsley.remote.js` is loaded:

```
<input name="q" type="text" data-parsley-remote data-parsley-remote-validator='mycustom' value="foo"
 />
[...]
<script href="parsley.remote.js"></script>
<script href="parsley.js"></script>
<script type="text/javascript">
$('[name="q"]').parsley()
  .addAsyncValidator('mycustom', function (xhr) {
    console.log(this.$element); // jQuery Object[ input[name="q"] ]

    return 404 === xhr.status;
  }, 'http://mycustomapiurl.ext');
</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 minimum 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