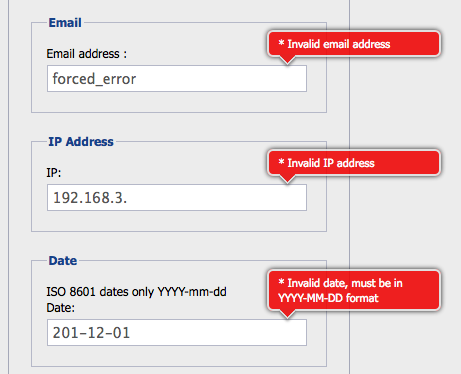
jQuery.validationEngine v2.6.1

Summary

**jQuery validation** engine is a Javascript plugin aimed at the validation of form fields in the browser (IE 6-8, Chrome, Firefox, Safari, Opera 10). The plugin provides visually appealing prompts that grab user attention on the subject matter.

Validations range from email, phone, and URL, to more complex calls such as ajax processing or custom javascript functions. Bundled with many locales, the error prompts can be translated into the language of your choice.



Documentation :

[Nicer documention](http://posabsolute.github.com/jQuery-Validation-Engine/)

[Release Notes](http://posabsolute.github.com/jQuery-Validation-Engine/releases.html)

Demo :

<http://www.position-relative.net/creation/formValidator/>

Installation

What's in the archive?

The archive holds, of course, the core library along with translations in different languages. It also comes with a set of demo pages and a simple ajax server (built in Java and php).

1. Unpack the archive
2. Include the script jquery.validationEngine.closure.js in your page
3. Pick the locale of the choice and include it in your page: jquery.validationEngine-XX.js
4. **Read this manual** and understand the API

Running the Demos

Most demos are fully functional by simply opening their respective HTML file. However, the Ajax demos require the use of Java6 to launch a lightweight http server.

1. Run the script runDemo.bat (Windows) or runDemo.sh (Unix) from the folder
2. Open a browser and point it at [http://localhost:9173](http://localhost:9173/)

Usage

References

First include jQuery on your page

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.js" type="text/

javascript"></script>

Include *jquery.validationEngine* and its locale

<script src="js/jquery.validationEngine-en.js" type="text/javascript" charset="utf-8"></script>

<script src="js/jquery.validationEngine.js" type="text/javascript" charset="utf-8"></script>

Finally include the desired theme

<link rel="stylesheet" href="css/validationEngine.jquery.css" type="text/css"/>

Field Validations

Validations are defined using the field's **class** attribute. Here are a few examples showing how it happens:

<input value="someone@nowhere.com" class="validate[required,custom[email]]" type="text" name="email" id="email" />

<input value="2010-12-01" class="validate[required,custom[date]]" type="text" name="date" id="date" />

<input value="too many spaces obviously" class="validate[required,custom[onlyLetterNumber]]" type="text" name="special" id="special" />

For more details about validators, please refer to the section below.

Experimental attribute data-validation-engine

We are currently in the process of replaceing the class attribute by something more standard, it should normally work but consider this feature in beta. 

Standard HTML5 attribute for error message

Customize error messages with data-errormessage and data-errormessage-\* attributes on the form elements. For example:

<input type="email" name="email" id="email" data-validation-engine="validate[required,custom[email]]"

data-errormessage-value-missing="Email is required!"

data-errormessage-custom-error="Let me give you a hint: someone@nowhere.com"

data-errormessage="This is the fall-back error message."/>

The following attribute's value will be loaded for the relative validation rule:

**data-errormessage-value-missing**

* required
* groupRequired
* condRequired

**data-errormessage-type-mismatch**

* past
* future
* dateRange
* dateTimeRange

**data-errormessage-pattern-mismatch**

* creditCard
* equals

**data-errormessage-range-underflow**

* minSize
* min
* minCheckbox

**data-errormessage-range-overflow**

* maxSize
* max
* maxCheckbox

**data-errormessage-custom-error**

* custom
* ajax
* funcCall

**data-errormessage**

* a generic fall-back error message

Per Field Prompt Direction

Prompt direction can be define using the field's **data** attribute. Here are a few examples showing how it happens:

<input value="http://" class="validate[required,custom[url]] text-input" type="text" name="url" id="url" data-prompt-position="topLeft" />

<input value="" class="validate[required] text-input" type="text" name="req" id="req" data-prompt-position="bottomLeft" />

<input value="too many spaces obviously" class="validate[required,custom[onlyLetterNumber]]" type="text" name="special" id="special" data-prompt-position="bottomRight" />

Prompt Position Adjustment

Prompt position can be adjusted by providing shiftX and shiftY with position type in the field's**data** attribute. Prompt will be placed in (defaultX+shiftX),(defaultY+shiftY) position instead of default for selected position type. Here are a few examples showing how it happens:

<input value="http://" class="validate[required,custom[url]] text-input" type="text" name="url" id="url" data-prompt-position="topLeft:70" />

<input value="" class="validate[required] text-input" type="text" name="req" id="req" data-prompt-position="bottomLeft:20,5" />

<input value="too many spaces obviously" class="validate[required,custom[onlyLetterNumber]]" type="text" name="special" id="special" data-prompt-position="bottomRight:-100,3" />

Instantiation

The validator is typically instantiated with a call in the following format, the plugin can only be instanciated on form elements:

$("#form.id").validationEngine();

Without any parameters, the init() and attach() methods are automatically called.

$("#form.id").validationEngine(action or options);

The method may take one or several parameters, either an action (and parameters) or a list of options to customize the behavior of the engine.

Here's a glimpse: say you have a form as such:

<form id="formID" method="post" action="submit.action">

<input value="2010-12-01" class="validate[required,custom[date]]" type="text" name="date" id="date" />

</form>

The code below will instantiate the validation engine and attach it to the form:

<script>

$(document).ready(function(){

$("#formID").validationEngine();

});

</script>

When using options, the default behavior is to only initialize ValidationEngine, so attachment needs to be done manually.

<script>

$(document).ready(function(){

$("#formID").validationEngine('attach', {promptPosition : "centerRight", scroll: false});

});

</script>

All calls to validationEngine() are chainable, so one can do the following:

$("#formID").validationEngine().css({border : "2px solid #000"});

Actions

attach

Attaches jQuery.validationEngine to form.submit and field.blur events.

$("#formID1").validationEngine('attach');

detach

Unregisters any bindings that may point to jQuery.validaitonEngine.

$("#formID1").validationEngine('detach');

validate

Validates a form or field, displays error prompts accordingly.  
Returns *true* if the form validates, *false* if it contains errors.

It is inversed for *fields*, it return false on validate and true on errors.

When using form validation with ajax, it returns *undefined* , the result is delivered asynchronously via function *options.onAjaxFormComplete*.

// form validation

alert( $("#formID1").validationEngine('validate') );

// field validation

alert( $("#emailInput").validationEngine('validate') );

showPrompt (promptText, type, promptPosition, showArrow)

Displays a prompt on a given element. Note that the prompt can be displayed on any element by providing an id.

The method takes four parameters: 1. the text of the prompt itself 2. a type which defines the visual look of the prompt: 'pass' (green), 'load' (black) anything else (red) 3. an optional position: either "topLeft", "topRight", "bottomLeft", "centerRight", "bottomRight". Defaults to *"topRight"* 4. an optional boolean which indicates if the prompt should display a directional arrow

<fieldset>

<legend id="legendid">Email</legend>

<a href="#" onclick="$('#legendid').validationEngine('showPrompt', 'This a custom msg', 'load')">Show prompt</a>

</fieldset>

hide

The hide method can be applied to a form or a field.  
It closes/hides error prompts.

// closes all form prompts

$('#formID1').validationEngine('hide');

// closes onle one prompt

$('#email1').validationEngine('hide');

hideAll

Closes/hides **all** error prompts on the page no matter what form they are attached to.

$('#formID1').validationEngine('hideAll');

updatePromptsPosition

Update the form prompts positions.

$("#formID").validationEngine("updatePromptsPosition")

hidePrompt & validateField

Deprecated and not part of the code base anymore.  
Use *hide* and *validate* instead.

Options

Options are typically passed to the init or attach action as a parameter.

$("#formID1").validationEngine({promptPosition : "centerRight", scroll: false});

$("#formID1").validationEngine('attach', {promptPosition : "centerRight", scroll: false});

validationEventTrigger

Name of the event triggering field validation, defaults to *blur*.

scroll

Determines if we should scroll the page to the first error, defaults to *true*.

binded

If set to true, it remove blur events and only validate on submit.

promptPosition

Where should the prompt show? Possible values are "topLeft", "topRight", "bottomLeft", "centerRight", "bottomRight". Defaults to *"topRight"*. Default position adjustment could also be provided.

showOneMessage

Only display the first incorrect validation message instead of normally stacking it. It will follows the validation hierarchy you used in the input and only show the first error.

ajaxFormValidation

If set to true, turns Ajax form validation logic on. Defaults to *false*. Form validation takes place when the validate() action is called or when the form is submitted.

ajaxFormValidationURL

If set, the ajax submit validation will use this url instead of the form action

ajaxFormValidationMethod

HTTP method used for ajax validation, defaults to 'get', can be set to 'post'

onBeforeAjaxFormValidation(form, options)

When ajaxFormValidation is turned on, this is the function that will be called before the asynchronous AJAX form validation call. May return false to stop the Ajax form validation

onAjaxFormComplete: function(status, form, errors, options)

When ajaxFormValidation is turned on, this function is used to asynchronously process the result of the validation. the status is a boolean. If true, the ajax call completed and all the server side form validations passed.

onValidationComplete

When defined, it stops by default the form from auto-submitting, and lets you handle the validation status via a function. You can also return true in this function and the form will be allowed to submit.

jQuery("#formID2").validationEngine('attach', {

onValidationComplete: function(form, status){

alert("The form status is: " +status+", it will never submit");

}

});

custom\_error\_messages

This is where custom messages for IDs, Classes, or validation types are stored.

Custom error messages areexclusive from one another.ID messages will be displayed instead of anything else; Class messages will only be used if there is no ID message, and only the first message found associated with one of the classes will be used; Global Validator messages will only be used if there are no Class messages or ID messages.

These custom messages are declared in this manner:

jQuery("#formID2").validationEngine({'custom\_error\_messages' : {

'#someId' : {

'required': {

'message': "This is a custom message that is only attached to the input with id 'someId' if it

has the validation of 'required'. This will always display, even if it has other

custom messages."

},

'custom[min]': {

'message': "This is a custom message that is only attached to the input with id 'someID' if it

has the validation of 'custom[min[someNumber]]'. This will always display, even if

it has other custom messages."

}

},

'.someClass': {

'equals': {

'message': "This is a custom message that is only attached to inputs that have the class of

'someClass' and the validation type of 'equals'. This will be displayed only on

inputs without an ID message."

}

},

'required' {

'message': "This is a custom message that replaces the normal error message for the validation

'required'. This only displays when there are no Class or ID messages."

}

}

});

focusFirstField

Specifies whether or not the first field in a form receives auto-focus after validation returns false. Default is set to *true*. If you want to disable the auto-focusing use:

$('#form').validationEngine('attach', {focusFirstField : false});

onSuccess

If set, this callback function will be called when all validations passed.

onFailure

If set, this callback function will be called when it found an error.

autoPositionUpdate

Auto update prompt position after window resize, disabled by default

autoHidePrompt

Determines if the prompt should hide itself automatically after a set period. Defaults to *false*.

autoHideDelay

Sets the number of ms that the prompt should appear for if autoHidePrompt is set to *true*. Defaults to *10000*.

Validators

Validators are encoded in the field's class attribute, as follows

required

Speaks for itself, fails if the element has no value. This validator can apply to pretty much any kind of input field.

<input value="" class="validate[required]" type="text" name="email" id="email" />

<input class="validate[required]" type="checkbox" id="agree" name="agree"/>

<select name="sport" id="sport" class="validate[required]" id="sport">

<option value="">Choose a sport</option>

<option value="option1">Tennis</option>

<option value="option2">Football</option>

<option value="option3">Golf</option>

</select>

groupRequired

At least one of the field of the group must be filled. It needs to be given a group name that is unique across the form.

<input value="" class="validate[groupRequired[payments]]" type="text" name="creditcard" id="creditcard" />

<input class="validate[groupRequired[payments]]" type="text" id="paypal" name="paypal"/>

condRequired

This makes the field required, but only if any of the referred fields has a value.

<input value="" type="text" name="creditcard" id="creditcard" />

<input class="validate[condRequired[creditcard]]" type="text" id="ccv" name="ccv"/>

custom[regex\_name]

Validates the element's value to a predefined list of regular expressions.

<input value="someone@nowhere.com" class="validate[required,custom[email]]" type="text" name="email" id="email" />

Please refer to the section *Custom Regex* for a list of available regular expressions.

custom[function\_name]

Validates the element's value to a predefined function included in the language file (compared to funCall that can be anywhere in your application),

<input value="someone@nowhere.com" class="validate[required,custom[requiredInFunction]]" type="text" name="email" id="email" />

Please refer to the section *Custom Regex* for a list of available regular expressions.

funcCall[methodName]

Validates a field using a third party function call. If a validation error occurs, the function must return an error message that will automatically show in the error prompt.

function checkHELLO(field, rules, i, options){

if (field.val() != "HELLO") {

// this allows the use of i18 for the error msgs

return options.allrules.validate2fields.alertText;

}

}

The following declaration will do

<input value="" class="validate[required,funcCall[checkHELLO]]" type="text" id="lastname" name="lastname" />

ajax[selector]

Delegates the validation to a server URL using an asynchronous Ajax request. The selector is used to identify a block of properties in the translation file, take the following for example.

\*\* The validation execution order is taken form the order you put them in the HTML, it is recommended to always put the ajax[] validation last. For example, the custom events might fail if you put ajax[] in the middle. Ajax[] works on submit since 2.6.

<input value="" class="validate[required,custom[onlyLetterNumber],maxSize[20],ajax[ajaxUserCall]] text-input" type="text" name="user" id="user" />

"ajaxUserCall": {

"url": "ajaxValidateFieldUser",

"extraData": "name=eric",

"extraDataDynamic": ['#user\_id', '#user\_email'],

"alertText": "\* This user is already taken",

"alertTextOk": "All good!",

"alertTextLoad": "\* Validating, please wait"

},

* url - is the remote restful service to call
* extraData - optional parameters to send
* extraDataDynamic - optional DOM id's that should have their values sent as parameters
* alertText - error prompt message if validation fails
* alertTextOk - optional prompt if validation succeeds (shows green)
* alertTextLoad - message displayed while the validation is being performed

This validator is explained in further details in the Ajax section.

equals[field.id]

Checks if the current field's value equals the value of the specified field.

min[float]

Validates when the field's value is less than, or equal to, the given parameter.

max[float]

Validates when the field's value is more than, or equal to, the given parameter.

minSize[integer]

Validates if the element content size (in characters) is more than, or equal to, the given *integer*. integer <= input.value.length

maxSize[integer]

Validates if the element content size (in characters) is less than, or equal to, the given *integer*. input.value.length <= integer

past[NOW, a date or another element's name]

Checks if the element's value (which is implicitly a date) is earlier than the given *date*. When "NOW" is used as a parameter, the date will be calculate in the browser. When a "#field name" is used ( The '#' is optional ), it will compare the element's value with another element's value within the same form. Note that this may be different from the server date. Dates use the ISO format YYYY-MM-DD

<input value="" class="validate[required,custom[date],past[now]]" type="text" id="birthdate" name="birthdate" />

<input value="" class="validate[required,custom[date],past[2010-01-01]]" type="text" id="appointment" name="appointment" />

<input value="" class="validate[required,custom[date],past[#appointment]]" type="text" id="restaurant" name="restaurant" />

<input value="" class="validate[required,custom[date],past[appointment]]" type="text" id="restaurant\_2" name="restaurant\_2" />

future[NOW, a date or another element's name]

Checks if the element's value (which is implicitly a date) is greater than the given *date*. When "NOW" is used as a parameter, the date will be calculate in the browser. When a "#field name" is used ( The '#' is optional ), it will compare the element's value with another element's value within the same form. Note that this may be different from the server date. Dates use the ISO format YYYY-MM-DD

<input value="" class="validate[required,custom[date],future[now]]" type="text" id="appointment" name="appointment" />

<input value="" class="validate[required,custom[date],future[#appointment]]" type="text" id="restaurant" name="restaurant" />

// a date in 2009

<input value="" class="validate[required,custom[date],future[2009-01-01],past[2009-12-31]]" type="text" id="d1" name="d1" />

minCheckbox[integer]

Validates when a minimum of *integer* checkboxes are selected. The validator uses a special naming convention to identify the checkboxes as part of a group.

The following example, enforces a minimum of two selected checkboxes

<input class="validate[minCheckbox[2]]" type="checkbox" name="group1" id="maxcheck1" value="5"/>

<input class="validate[minCheckbox[2]]" type="checkbox" name="group1" id="maxcheck2" value="3"/>

<input class="validate[minCheckbox[2]]" type="checkbox" name="group1" id="maxcheck3" value="9"/>

Note how the input.name is identical across the fields.

maxCheckbox[integer]

Same as above but limits the maximum number of selected check boxes.

creditCard

Validates that a credit card number is at least theoretically valid, according the to the [Luhn checksum algorithm](http://en.wikipedia.org/wiki/Luhn_algorithm), but not whether the specific card number is active with a bank, etc.

Selectors

We've introduced the notion of selectors without giving many details about them: A selector is a string which is used as a key to match properties in the translation files. Take the following example:

"onlyNumber": {

"regex": /^[0-9\ ]+$/,

"alertText": "\* Numbers only"

},

"ajaxUserCall": {

"url": "ajaxValidateFieldUser",

"extraData": "name=eric",

"alertText": "\* This user is already taken",

"alertTextOk": " \* User is valid",

"alertTextLoad": "\* Validating, please wait"

},

"validate2fields": {

"alertText": "\* Please input HELLO"

}

onlyNumber, onlyLetter and validate2fields are all selectors. jQuery.validationEngine comes with a standard set but you are welcome to add you own to define AJAX backend services, error messages and/or new regular expressions.

The ValidationEngine with a datepicker

Using a datepicker with the engine is problematic because the validation is bound to the blur event. since we lose the focus before any data is entered in the field it creates a weird bug. Fortunately we implemented a fix that uses a delay during the datepicker binding.

To use this mode you need to add the class *datepicker* to your input, like this:

<input type="text" id="req" name="req" class="validate[required] text-input datepicker" value="">

Ajax Protocol

The ajax validator takes a selector as an attribute. the selector points to a structure that defines the URL to call, the different messages to display and any extra parameters to add on the URL (when applicable). Please refer to the **ajax[selector]** description for more details.

Ajax validation comes in two flavors:

1. Field Ajax validations, which take place when the user inputs a value in a field and moves away.
2. Form Ajax validation, which takes place when the form is submitted or when the validate() action is called.

Both options are optional.

<input value="" class="validate[required,ajax[ajaxUserCall]] text-input" type="text" name="user" id="user" />

You can see a tutorial that makes the use of php here: <http://www.position-absolute.com/articles/using-form-ajax-validation-with-the-jquery-validation-engine-plugin/>

Field ajax validation

Protocol

The client sends the fieldId and the fieldValue as a GET request to the server url.

Client calls url?fieldId=id1&fieldValue=value1 ==> Server

Server responds with an array: [fieldid, status, errorMsg].

Client receives <== ["id1", boolean, errorMsg] Server

* fieldid is the name (id) of the field
* status is the result of the validation, true if it passes, false if it fails
* errorMsg is an optional error string (or a selector) to the prompt text. If no error msg is returned, the prompt message is expected to be part of the rule with key "alertText" or "alertTextOk" (see the structure of the translation file)

Form ajax validation

Protocol

The client sends the form fields and values as a GET request to the form.action url.

Client calls url?fieldId=id1&fieldValue=value1&...etc ==> Server (form.action)

Server responds with an *array of arrays*: [fieldid, status, errorMsg].

* fieldid is the name (id) of the field
* status is the result of the validation, true if it passes, false if it fails
* errorMsg is an error string (or a selector) to the prompt text

Client receives <== [["id1", boolean,"errorMsg"],["id2", false, "there is an error "],["id3", true, "this field is good"]]

Note that normally errors (status=false) are returned from the server. However you may also decide to return an entry with a status=true in which case the errorMsg will show as a green prompt.

Validation URL

By default the engine use the form action to validate the form, you can however set a default url using:

\*\*ajaxFormValidationURL

Callbacks

Since the form validation is asynchronously delegated to the form action, we provide two callback methods:

**onBeforeAjaxFormValidation(form, options)** is called before the ajax form validation call, it may return false to stop the request

**onAjaxFormComplete: function(form, status, json\_response\_from\_server, options)** is called after the ajax form validation call

Custom Regex

jQuery.validationEngine comes with a lot of predefined expressions. Regex validation rules are specified as follows:

<input value="" class="validate[custom[email]]" type="text" name="email" id="email" />

Note that the selector identifies a given regular expression in the translation file, but also its associated error prompt messages and optional green prompt message.

phone

a typical phone number with an optional country code and extension. Note that the validation is**relaxed**, please add extra validations for your specific country.

49-4312 / 777 777

+1 (305) 613-0958 x101

(305) 613 09 58 ext 101

3056130958

+33 1 47 37 62 24 extension 3

(016977) 1234

04312 - 777 777

91-12345-12345

+58 295416 7216

url

Matches a URL such as [http://myserver.com](http://myserver.com/), [https://www.crionics.com](https://www.crionics.com/) or ftp://myserver.ws

email

Easy, an email : username@hostname.com

date

An ISO date, YYYY-MM-DD

number

Floating points with an optional sign. ie. -143.22 or .77 but also +234,23

integer

Integers with an optional sign. ie. -635 +2201 738

ipv4

An IP address (v4) ie. 127.0.0.1

onlyNumberSp

Only numbers and spaces characters

onlyLetterSp

Only letters and space characters

onlyLetterNumber

Only letters and numbers, no space

Position fixed and overflow scroll

Before 2.5.3 some options were needed to use the engine in a div with overflow scroll or position fixed, now you just have to set position relative on the form and you are good to go.

Placeholders

The engine support by default placeholders when a field is required. use the attribute data-validation-placeholder in the input to define it.



Hooks

The plugin provides some hooks using jQuery bind functionality.

* jqv.form.validating : Trigger when the form is submitted and before it starts the validation process
* jqv.field.result(event, field, errorFound, prompText) : Triggers when a field is validated with the result.
* jqv.form.result(event, errorFound) : Triggers when a form is validated with the result

An example of binding a custom function to these events would be:

$("#formID").bind("jqv.form.result", function(event, errorFound) {

if(errorFound)

alert("There is a problem with your form");

});

Customizations

What would a good library be without customization?

Adding regular expressions

Adding new regular expressions is easy: open your translation file and add a new entry to the list

"onlyLetter": {

"regex": /^[a-zA-Z\ \']+$/,

"alertText": "\* Letters only"

},

* "onlyLetter" is a sample selector name
* "regex" is a javascript regular expression
* "alertText" is the message to display when the validation fails

You can now use the new regular expression as such

<input type="text" id="myid" name="myid" class="validation[custom[onlyLetter]]"/>

Don't forget to contribute!

Customizing the look and feel

Edit the file *validationEngine.jquery.css* and customize the stylesheet to your liking. it's trivial if you know CSS.

Adding more locales

You can easily add a locale by taking *jquery.validationEngine-en.js* as an example. Feel free to share the translation ;-)

Changing defaults options globally

You can, for example, disable the scrolling globally by using $.validationEngine.defaults.scroll = false.

This need to be added before the initialization, one good way to handle this would be to add your settings in a file.

<script src="js/jquery.validationEngine-en.js" type="text/javascript" charset="utf-8"></script>

<script src="js/jquery.validationEngine.js" type="text/javascript" charset="utf-8"></script>

<script src="js/jquery.validationEngine-settings.js" type="text/javascript" charset="utf-8"></script>

Using the validationEngine with modal & dialog plugins

You can have more information about implementing the engine with modal views here: [<http://www.position-absolute.com/articles/using-the-jquery-validation-engine-with-modal-plugins/>]

Rules of thumb

* field.id is **unique** across the page
* From 2.2.4 and up, jquery 1.6+ is required because of prop()
* for simplicity and consistency field.id and field.name should match (except with minCheckbox and maxCheckbox validators)
* spaces or special chars should be avoided in field.id or field.name
* use lower case for input.type ie. *text, password, textarea, checkbox, radio*
* validators are evaluated from left to right, use the Ajax validator last e.g. validate[custom[onlyLetter],length[0,100],**ajax[ajaxNameCall]**]
* please use only one Ajax validator per field!
* JSON services should live on the same server (or you will get into browser security issues)
* in a perfect RESTful world, http **GET** is used to *READ* data, http **POST** is used to *WRITE* data: which translates into -> Ajax validations should use GET, the actual form post should use a POST request.

Contribution

Contributions are always welcome, please follow these steps to submit your changes:

1. Install git from <http://git-scm.com/>
2. Create a github account on [https://github.com](https://github.com/)
3. Set up your git ssh key using these instructions <http://help.github.com/set-up-git-redirect>
4. Open the jQuery Validation Engine project home page on github on<https://github.com/posabsolute/jQuery-Validation-Engine>
5. Click the "Fork" button, this will get you to a new page: your own copy of the code.
6. Copy the SSH URL at the top of the page and clone the repository on your local machine

git clone git@github.com:your-username/jQuery-Validation-Engine.git my-jqv-repo

1. Create a branch and switch to it
2. cd my-jqv-repo
3. git branch mynewfeature-patch

git checkout mynewfeature-patch

1. Apply your changes, then commit using a meaningful comment, that's the comment everybody will see!
2. git add .

git commit -m "Fixing issue 157, blablabla"

1. Push the changes back to github (under a different branch, here myfeature-patch)

git push origin mynewfeature-patch

1. Open your forked repository on github at <https://github.com/your-username/jQuery-Validation-Engine>
2. Click "Switch Branches" and select your branch (mynewfeature-patch)
3. Click "Pull Request"
4. Submit your pull request to JQV Developers

Support

We offer limited support on <http://www.stackoverflow.com/>

Use the tag jQuery-Validation-Engine

License

Licensed under the MIT License

Authors

Copyright(c) 2011 [Cedric Dugas](https://github.com/posabsolute) [http://www.position-absolute.com](http://www.position-absolute.com/)

v2.0 Rewrite by [Olivier Refalo](https://github.com/orefalo) [http://www.crionics.com](http://www.crionics.com/)