* Jump To … +  
    
  [<<< back to documentation](http://docs.google.com/index.html)  [base.js](http://docs.google.com/base.html)   [constraint.js](http://docs.google.com/constraint.html)   [defaults.js](http://docs.google.com/defaults.html)   [factory.js](http://docs.google.com/factory.html)   [field.js](http://docs.google.com/field.html)   [form.js](http://docs.google.com/form.html)   [main.js](http://docs.google.com/main.html)   [multiple.js](http://docs.google.com/multiple.html)   [pubsub.js](http://docs.google.com/pubsub.html)   [remote.js](http://docs.google.com/remote.html)   [ui.js](http://docs.google.com/ui.html)   [utils.js](http://docs.google.com/utils.html)   [validator.js](http://docs.google.com/validator.html)   [validator\_registry.js](http://docs.google.com/validator_registry.html)

# validator\_registry.js

* [¶](#gjdgxs)  
  import $ from 'jquery';  
  import Utils from './utils';  
  import Defaults from './defaults';  
  import Validator from './validator';  
    
  var ValidatorRegistry = function (validators, catalog) {  
   this.\_\_class\_\_ = 'ValidatorRegistry';
* [¶](#30j0zll)  
  Default Parsley locale is en  
   this.locale = 'en';  
    
   this.init(validators || {}, catalog || {});  
  };  
    
  var typeTesters = {  
   email: /^((([a-zA-Z]|\d|[!#\$%&'\\*\+\-\/=\?\^\_`{\|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])+(\.([a-zA-Z]|\d|[!#\$%&'\\*\+\-\/=\?\^\_`{\|}~]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])+)\*)|((\x22)((((\x20|\x09)\*(\x0d\x0a))?(\x20|\x09)+)?(([\x01-\x08\x0b\x0c\x0e-\x1f\x7f]|\x21|[\x23-\x5b]|[\x5d-\x7e]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(\\([\x01-\x09\x0b\x0c\x0d-\x7f]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF]))))\*(((\x20|\x09)\*(\x0d\x0a))?(\x20|\x09)+)?(\x22)))@((([a-zA-Z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-zA-Z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-zA-Z]|\d|-|\_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])\*([a-zA-Z]|\d|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))\.)+(([a-zA-Z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])|(([a-zA-Z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])([a-zA-Z]|\d|-|\_|~|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])\*([a-zA-Z]|[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF])))$/,
* [¶](#1fob9te)  
  Follow <https://www.w3.org/TR/html5/infrastructure.html#floating-point-numbers>  
   number: /^-?(\d\*\.)?\d+(e[-+]?\d+)?$/i,  
    
   integer: /^-?\d+$/,  
    
   digits: /^\d+$/,  
    
   alphanum: /^\w+$/i,  
    
   date: {  
   test: value => Utils.parse.date(value) !== null  
   },  
    
   url: new RegExp(  
   "^" +
* [¶](#3znysh7)  
  protocol identifier  
   "(?:(?:https?|ftp)://)?" + // \*\* mod: make scheme optional
* [¶](#2et92p0)  
  user:pass authentication  
   "(?:\\S+(?::\\S\*)?@)?" +  
   "(?:" +
* [¶](#tyjcwt)  
  IP address exclusion private & local networks “(?!(?:10|127)(?:\.\d{1,3}){3})” + //  **mod: allow local networks “(?!(?:169\.254|192\.168)(?:\.\d{1,3}){2})” + //**  mod: allow local networks “(?!172\.(?:1[6-9]|2\d|3[0-1])(?:\.\d{1,3}){2})” + // \*\* mod: allow local networks IP address dotted notation octets excludes loopback network 0.0.0.0 excludes reserved space >= 224.0.0.0 excludes network & broacast addresses (first & last IP address of each class)  
   "(?:[1-9]\\d?|1\\d\\d|2[01]\\d|22[0-3])" +  
   "(?:\\.(?:1?\\d{1,2}|2[0-4]\\d|25[0-5])){2}" +  
   "(?:\\.(?:[1-9]\\d?|1\\d\\d|2[0-4]\\d|25[0-4]))" +  
   "|" +
* [¶](#3dy6vkm)  
  host name  
   "(?:(?:[a-zA-Z\\u00a1-\\uffff0-9]-\*)\*[a-zA-Z\\u00a1-\\uffff0-9]+)" +
* [¶](#1t3h5sf)  
  domain name  
   "(?:\\.(?:[a-zA-Z\\u00a1-\\uffff0-9]-\*)\*[a-zA-Z\\u00a1-\\uffff0-9]+)\*" +
* [¶](#4d34og8)  
  TLD identifier  
   "(?:\\.(?:[a-zA-Z\\u00a1-\\uffff]{2,}))" +  
   ")" +
* [¶](#2s8eyo1)  
  port number  
   "(?::\\d{2,5})?" +
* [¶](#17dp8vu)  
  resource path  
   "(?:/\\S\*)?" +  
   "$"  
   )  
  };  
  typeTesters.range = typeTesters.number;
* [¶](#3rdcrjn)  
  See <http://stackoverflow.com/a/10454560/8279>  
  var decimalPlaces = num => {  
   var match = ('' + num).match(/(?:\.(\d+))?(?:[eE]([+-]?\d+))?$/);  
   if (!match) { return 0; }  
   return Math.max(  
   0,
* [¶](#26in1rg)  
  Number of digits right of decimal point.  
   (match[1] ? match[1].length : 0) -
* [¶](#lnxbz9)  
  Adjust for scientific notation.  
   (match[2] ? +match[2] : 0));  
  };
* [¶](#35nkun2)  
  parseArguments(‘number’, [‘1’, ‘2’]) => [1, 2]  
  let parseArguments = (type, args) => args.map(Utils.parse[type]);
* [¶](#1ksv4uv)  
  operatorToValidator returns a validating function for an operator function, applied to the given type  
  let operatorToValidator = (type, operator) => {  
   return (value, ...requirementsAndInput) => {  
   requirementsAndInput.pop(); // Get rid of `input` argument  
   return operator(value, ...parseArguments(type, requirementsAndInput));  
   };  
  };  
    
  let comparisonOperator = operator => ({  
   validateDate: operatorToValidator('date', operator),  
   validateNumber: operatorToValidator('number', operator),  
   requirementType: operator.length <= 2 ? 'string' : ['string', 'string'], // Support operators with a 1 or 2 requirement(s)  
   priority: 30  
  });  
    
  ValidatorRegistry.prototype = {  
   init: function (validators, catalog) {  
   this.catalog = catalog;
* [¶](#44sinio)  
  Copy prototype’s validators:  
   this.validators = Object.assign({}, this.validators);  
    
   for (var name in validators)  
   this.addValidator(name, validators[name].fn, validators[name].priority);  
    
   window.Parsley.trigger('parsley:validator:init');  
   },
* [¶](#2jxsxqh)  
  Set new messages locale if we have dictionary loaded in ParsleyConfig.i18n  
   setLocale: function (locale) {  
   if ('undefined' === typeof this.catalog[locale])  
   throw new Error(locale + ' is not available in the catalog');  
    
   this.locale = locale;  
    
   return this;  
   },
* [¶](#z337ya)  
  Add a new messages catalog for a given locale. Set locale for this catalog if set === true  
   addCatalog: function (locale, messages, set) {  
   if ('object' === typeof messages)  
   this.catalog[locale] = messages;  
    
   if (true === set)  
   return this.setLocale(locale);  
    
   return this;  
   },
* [¶](#3j2qqm3)  
  Add a specific message for a given constraint in a given locale  
   addMessage: function (locale, name, message) {  
   if ('undefined' === typeof this.catalog[locale])  
   this.catalog[locale] = {};  
    
   this.catalog[locale][name] = message;  
    
   return this;  
   },
* [¶](#1y810tw)  
  Add messages for a given locale  
   addMessages: function (locale, nameMessageObject) {  
   for (var name in nameMessageObject)  
   this.addMessage(locale, name, nameMessageObject[name]);  
    
   return this;  
   },
* [¶](#4i7ojhp)  
  Add a new validator  
  addValidator(‘custom’, { requirementType: [‘integer’, ‘integer’], validateString: function(value, from, to) {}, priority: 22, messages: { en: “Hey, that’s no good”, fr: “Aye aye, pas bon du tout”, } })  
  Old API was addValidator(name, function, priority)  
   addValidator: function (name, arg1, arg2) {  
   if (this.validators[name])  
   Utils.warn('Validator "' + name + '" is already defined.');  
   else if (Defaults.hasOwnProperty(name)) {  
   Utils.warn('"' + name + '" is a restricted keyword and is not a valid validator name.');  
   return;  
   }  
   return this.\_setValidator(...arguments);  
   },  
    
   hasValidator: function (name) {  
   return !!this.validators[name];  
   },  
    
   updateValidator: function (name, arg1, arg2) {  
   if (!this.validators[name]) {  
   Utils.warn('Validator "' + name + '" is not already defined.');  
   return this.addValidator(...arguments);  
   }  
   return this.\_setValidator(...arguments);  
   },  
    
   removeValidator: function (name) {  
   if (!this.validators[name])  
   Utils.warn('Validator "' + name + '" is not defined.');  
    
   delete this.validators[name];  
    
   return this;  
   },  
    
   \_setValidator: function (name, validator, priority) {  
   if ('object' !== typeof validator) {
* [¶](#2xcytpi)  
  Old style validator, with fn and priority  
   validator = {  
   fn: validator,  
   priority: priority  
   };  
   }  
   if (!validator.validate) {  
   validator = new Validator(validator);  
   }  
   this.validators[name] = validator;  
    
   for (var locale in validator.messages || {})  
   this.addMessage(locale, name, validator.messages[locale]);  
    
   return this;  
   },  
    
   getErrorMessage: function (constraint) {  
   var message;
* [¶](#1ci93xb)  
  Type constraints are a bit different, we have to match their requirements too to find right error message  
   if ('type' === constraint.name) {  
   var typeMessages = this.catalog[this.locale][constraint.name] || {};  
   message = typeMessages[constraint.requirements];  
   } else  
   message = this.formatMessage(this.catalog[this.locale][constraint.name], constraint.requirements);  
    
   return message || this.catalog[this.locale].defaultMessage || this.catalog.en.defaultMessage;  
   },
* [¶](#3whwml4)  
  Kind of light sprintf() implementation  
   formatMessage: function (string, parameters) {  
   if ('object' === typeof parameters) {  
   for (var i in parameters)  
   string = this.formatMessage(string, parameters[i]);  
    
   return string;  
   }  
    
   return 'string' === typeof string ? string.replace(/%s/i, parameters) : '';  
   },
* [¶](#2bn6wsx)  
  Here is the Parsley default validators list. A validator is an object with the following key values:
  + priority: an integer
  + requirement: ‘string’ (default), ‘integer’, ‘number’, ‘regexp’ or an Array of these
  + validateString, validateMultiple, validateNumber: functions returning true, false or a promise Alternatively, a validator can be a function that returns such an object

validators: {  
 notblank: {  
 validateString: function(value) {  
 return /\S/.test(value);  
 },  
 priority: 2  
 },  
 required: {  
 validateMultiple: function(values) {  
 return values.length > 0;  
 },  
 validateString: function(value) {  
 return /\S/.test(value);  
 },  
 priority: 512  
 },  
 type: {  
 validateString: function(value, type, {step = 'any', base = 0} = {}) {  
 var tester = typeTesters[type];  
 if (!tester) {  
 throw new Error('validator type `' + type + '` is not supported');  
 }  
 if (!tester.test(value))  
 return false;  
 if ('number' === type) {  
 if (!/^any$/i.test(step || '')) {  
 var nb = Number(value);  
 var decimals = Math.max(decimalPlaces(step), decimalPlaces(base));  
 if (decimalPlaces(nb) > decimals) // Value can't have too many decimals  
 return false;

* [¶](#qsh70q)  
  Be careful of rounding errors by using integers.  
   var toInt = f => Math.round(f \* Math.pow(10, decimals));  
   if ((toInt(nb) - toInt(base)) % toInt(step) != 0)  
   return false;  
   }  
   }  
   return true;  
   },  
   requirementType: {  
   '': 'string',  
   step: 'string',  
   base: 'number'  
   },  
   priority: 256  
   },  
   pattern: {  
   validateString: function(value, regexp) {  
   return regexp.test(value);  
   },  
   requirementType: 'regexp',  
   priority: 64  
   },  
   minlength: {  
   validateString: function (value, requirement) {  
   return value.length >= requirement;  
   },  
   requirementType: 'integer',  
   priority: 30  
   },  
   maxlength: {  
   validateString: function (value, requirement) {  
   return value.length <= requirement;  
   },  
   requirementType: 'integer',  
   priority: 30  
   },  
   length: {  
   validateString: function (value, min, max) {  
   return value.length >= min && value.length <= max;  
   },  
   requirementType: ['integer', 'integer'],  
   priority: 30  
   },  
   mincheck: {  
   validateMultiple: function (values, requirement) {  
   return values.length >= requirement;  
   },  
   requirementType: 'integer',  
   priority: 30  
   },  
   maxcheck: {  
   validateMultiple: function (values, requirement) {  
   return values.length <= requirement;  
   },  
   requirementType: 'integer',  
   priority: 30  
   },  
   check: {  
   validateMultiple: function (values, min, max) {  
   return values.length >= min && values.length <= max;  
   },  
   requirementType: ['integer', 'integer'],  
   priority: 30  
   },  
   min: comparisonOperator((value, requirement) => value >= requirement),  
   max: comparisonOperator((value, requirement) => value <= requirement),  
   range: comparisonOperator((value, min, max) => value >= min && value <= max),  
   equalto: {  
   validateString: function (value, refOrValue) {  
   var $reference = $(refOrValue);  
   if ($reference.length)  
   return value === $reference.val();  
   else  
   return value === refOrValue;  
   },  
   priority: 256  
   }  
   }  
  };  
    
  export default ValidatorRegistry;