* 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.js

* [¶](#gjdgxs)  
  import $ from 'jquery';  
  import Utils from './utils';  
    
  var convertArrayRequirement = function(string, length) {  
   var m = string.match(/^\s\*\[(.\*)\]\s\*$/);  
   if (!m)  
   throw 'Requirement is not an array: "' + string + '"';  
   var values = m[1].split(',').map(Utils.trimString);  
   if (values.length !== length)  
   throw 'Requirement has ' + values.length + ' values when ' + length + ' are needed';  
   return values;  
  };  
    
  var convertExtraOptionRequirement = function(requirementSpec, string, extraOptionReader) {  
   var main = null;  
   var extra = {};  
   for (var key in requirementSpec) {  
   if (key) {  
   var value = extraOptionReader(key);  
   if ('string' === typeof value)  
   value = Utils.parseRequirement(requirementSpec[key], value);  
   extra[key] = value;  
   } else {  
   main = Utils.parseRequirement(requirementSpec[key], string);  
   }  
   }  
   return [main, extra];  
  };
* [¶](#30j0zll)  
  A Validator needs to implement the methods validate and parseRequirements  
  var Validator = function(spec) {  
   $.extend(true, this, spec);  
  };  
    
  Validator.prototype = {
* [¶](#1fob9te)  
  Returns true iff the given value is valid according the given requirements.  
   validate: function(value, requirementFirstArg) {  
   if (this.fn) { // Legacy style validator  
    
   if (arguments.length > 3) // If more args then value, requirement, instance...  
   requirementFirstArg = [].slice.call(arguments, 1, -1); // Skip first arg (value) and last (instance), combining the rest  
   return this.fn(value, requirementFirstArg);  
   }  
    
   if (Array.isArray(value)) {  
   if (!this.validateMultiple)  
   throw 'Validator `' + this.name + '` does not handle multiple values';  
   return this.validateMultiple(...arguments);  
   } else {  
   let instance = arguments[arguments.length - 1];  
   if (this.validateDate && instance.\_isDateInput()) {  
   arguments[0] = Utils.parse.date(arguments[0]);  
   if (arguments[0] === null)  
   return false;  
   return this.validateDate(...arguments);  
   }  
   if (this.validateNumber) {  
   if (isNaN(value))  
   return false;  
   arguments[0] = parseFloat(arguments[0]);  
   return this.validateNumber(...arguments);  
   }  
   if (this.validateString) {  
   return this.validateString(...arguments);  
   }  
   throw 'Validator `' + this.name + '` only handles multiple values';  
   }  
   },
* [¶](#3znysh7)  
  Parses requirements into an array of arguments, according to this.requirementType  
   parseRequirements: function(requirements, extraOptionReader) {  
   if ('string' !== typeof requirements) {
* [¶](#2et92p0)  
  Assume requirement already parsed but make sure we return an array  
   return Array.isArray(requirements) ? requirements : [requirements];  
   }  
   var type = this.requirementType;  
   if (Array.isArray(type)) {  
   var values = convertArrayRequirement(requirements, type.length);  
   for (var i = 0; i < values.length; i++)  
   values[i] = Utils.parseRequirement(type[i], values[i]);  
   return values;  
   } else if ($.isPlainObject(type)) {  
   return convertExtraOptionRequirement(type, requirements, extraOptionReader);  
   } else {  
   return [Utils.parseRequirement(type, requirements)];  
   }  
   },
* [¶](#tyjcwt)  
  Defaults:  
   requirementType: 'string',  
    
   priority: 2  
    
  };  
    
  export default Validator;