* 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)

# field.js

* [¶](#gjdgxs)  
  import $ from 'jquery';  
  import Constraint from './constraint';  
  import UI from './ui';  
  import Utils from './utils';  
    
  var Field = function (field, domOptions, options, parsleyFormInstance) {  
   this.\_\_class\_\_ = 'Field';  
    
   this.element = field;  
   this.$element = $(field);
* [¶](#30j0zll)  
  Set parent if we have one  
   if ('undefined' !== typeof parsleyFormInstance) {  
   this.parent = parsleyFormInstance;  
   }  
    
   this.options = options;  
   this.domOptions = domOptions;
* [¶](#1fob9te)  
  Initialize some properties  
   this.constraints = [];  
   this.constraintsByName = {};  
   this.validationResult = true;
* [¶](#3znysh7)  
  Bind constraints  
   this.\_bindConstraints();  
  };  
    
  var statusMapping = {pending: null, resolved: true, rejected: false};  
    
  Field.prototype = {

# [¶](#2et92p0) Public API Validate field and trigger some events for mainly UI @returns true, an array of the validators that failed, or null if validation is not finished. Prefer using whenValidate validate: function (options) { if (arguments.length >= 1 && !$.isPlainObject(options)) { Utils.warnOnce('Calling validate on a parsley field without passing arguments as an object is deprecated.'); options = {options}; } var promise = this.whenValidate(options); if (!promise) // If excluded with `group` option return true; switch (promise.state()) { case 'pending': return null; case 'resolved': return true; case 'rejected': return this.validationResult; } },

* [¶](#tyjcwt)  
  Validate field and trigger some events for mainly UI @returns a promise that succeeds only when all validations do or undefined if field is not in the given group.  
   whenValidate: function ({force, group} = {}) {
* [¶](#3dy6vkm)  
  do not validate a field if not the same as given validation group  
   this.refresh();  
   if (group && !this.\_isInGroup(group))  
   return;  
    
   this.value = this.getValue();
* [¶](#1t3h5sf)  
  Field Validate event. this.value could be altered for custom needs  
   this.\_trigger('validate');  
    
   return this.whenValid({force, value: this.value, \_refreshed: true})  
   .always(() => { this.\_reflowUI(); })  
   .done(() => { this.\_trigger('success'); })  
   .fail(() => { this.\_trigger('error'); })  
   .always(() => { this.\_trigger('validated'); })  
   .pipe(...this.\_pipeAccordingToValidationResult());  
   },  
    
   hasConstraints: function () {  
   return 0 !== this.constraints.length;  
   },
* [¶](#4d34og8)  
  An empty optional field does not need validation  
   needsValidation: function (value) {  
   if ('undefined' === typeof value)  
   value = this.getValue();
* [¶](#2s8eyo1)  
  If a field is empty and not required, it is valid Except if data-parsley-validate-if-empty explicitely added, useful for some custom validators  
   if (!value.length && !this.\_isRequired() && 'undefined' === typeof this.options.validateIfEmpty)  
   return false;  
    
   return true;  
   },  
    
   \_isInGroup: function (group) {  
   if (Array.isArray(this.options.group))  
   return -1 !== $.inArray(group, this.options.group);  
   return this.options.group === group;  
   },
* [¶](#17dp8vu)  
  Just validate field. Do not trigger any event. Returns true iff all constraints pass, false if there are failures, or null if the result can not be determined yet (depends on a promise) See also whenValid.  
   isValid: function (options) {  
   if (arguments.length >= 1 && !$.isPlainObject(options)) {  
   Utils.warnOnce('Calling isValid on a parsley field without passing arguments as an object is deprecated.');  
   var [force, value] = arguments;  
   options = {force, value};  
   }  
   var promise = this.whenValid(options);  
   if (!promise) // Excluded via `group`  
   return true;  
   return statusMapping[promise.state()];  
   },
* [¶](#3rdcrjn)  
  Just validate field. Do not trigger any event. @returns a promise that succeeds only when all validations do or undefined if the field is not in the given group. The argument force will force validation of empty fields. If a value is given, it will be validated instead of the value of the input.  
   whenValid: function ({force = false, value, group, \_refreshed} = {}) {
* [¶](#26in1rg)  
  Recompute options and rebind constraints to have latest changes  
   if (!\_refreshed)  
   this.refresh();
* [¶](#lnxbz9)  
  do not validate a field if not the same as given validation group  
   if (group && !this.\_isInGroup(group))  
   return;  
    
   this.validationResult = true;
* [¶](#35nkun2)  
  A field without constraint is valid  
   if (!this.hasConstraints())  
   return $.when();
* [¶](#1ksv4uv)  
  Value could be passed as argument, needed to add more power to ‘field:validate’  
   if ('undefined' === typeof value || null === value)  
   value = this.getValue();  
    
   if (!this.needsValidation(value) && true !== force)  
   return $.when();  
    
   var groupedConstraints = this.\_getGroupedConstraints();  
   var promises = [];  
   $.each(groupedConstraints, (\_, constraints) => {
* [¶](#44sinio)  
  Process one group of constraints at a time, we validate the constraints and combine the promises together.  
   var promise = Utils.all(  
   $.map(constraints, constraint => this.\_validateConstraint(value, constraint))  
   );  
   promises.push(promise);  
   if (promise.state() === 'rejected')  
   return false; // Interrupt processing if a group has already failed  
   });  
   return Utils.all(promises);  
   },
* [¶](#2jxsxqh)  
  @returns a promise  
   \_validateConstraint: function(value, constraint) {  
   var result = constraint.validate(value, this);
* [¶](#z337ya)  
  Map false to a failed promise  
   if (false === result)  
   result = $.Deferred().reject();
* [¶](#3j2qqm3)  
  Make sure we return a promise and that we record failures  
   return Utils.all([result]).fail(errorMessage => {  
   if (!(this.validationResult instanceof Array))  
   this.validationResult = [];  
   this.validationResult.push({  
   assert: constraint,  
   errorMessage: 'string' === typeof errorMessage && errorMessage  
   });  
   });  
   },
* [¶](#1y810tw)  
  @returns Parsley field computed value that could be overrided or configured in DOM  
   getValue: function () {  
   var value;
* [¶](#4i7ojhp)  
  Value could be overriden in DOM or with explicit options  
   if ('function' === typeof this.options.value)  
   value = this.options.value(this);  
   else if ('undefined' !== typeof this.options.value)  
   value = this.options.value;  
   else  
   value = this.$element.val();
* [¶](#2xcytpi)  
  Handle wrong DOM or configurations  
   if ('undefined' === typeof value || null === value)  
   return '';  
    
   return this.\_handleWhitespace(value);  
   },
* [¶](#1ci93xb)  
  Reset UI  
   reset: function () {  
   this.\_resetUI();  
   return this.\_trigger('reset');  
   },
* [¶](#3whwml4)  
  Destroy Parsley instance (+ UI)  
   destroy: function () {
* [¶](#2bn6wsx)  
  Field case: emit destroy event to clean UI and then destroy stored instance  
   this.\_destroyUI();  
   this.$element.removeData('Parsley');  
   this.$element.removeData('FieldMultiple');  
   this.\_trigger('destroy');  
   },
* [¶](#qsh70q)  
  Actualize options and rebind constraints  
   refresh: function () {  
   this.\_refreshConstraints();  
   return this;  
   },  
    
   \_refreshConstraints: function () {  
   return this.actualizeOptions().\_bindConstraints();  
   },  
    
   refreshConstraints: function() {  
   Utils.warnOnce("Parsley's refreshConstraints is deprecated. Please use refresh");  
   return this.refresh();  
   },  
    
   /\*\*  
   \* Add a new constraint to a field  
   \*  
   \* @param {String} name  
   \* @param {Mixed} requirements optional  
   \* @param {Number} priority optional  
   \* @param {Boolean} isDomConstraint optional  
   \*/  
   addConstraint: function (name, requirements, priority, isDomConstraint) {  
    
   if (window.Parsley.\_validatorRegistry.validators[name]) {  
   var constraint = new Constraint(this, name, requirements, priority, isDomConstraint);
* [¶](#3as4poj)  
  if constraint already exist, delete it and push new version  
   if ('undefined' !== this.constraintsByName[constraint.name])  
   this.removeConstraint(constraint.name);  
    
   this.constraints.push(constraint);  
   this.constraintsByName[constraint.name] = constraint;  
   }  
    
   return this;  
   },
* [¶](#1pxezwc)  
  Remove a constraint  
   removeConstraint: function (name) {  
   for (var i = 0; i < this.constraints.length; i++)  
   if (name === this.constraints[i].name) {  
   this.constraints.splice(i, 1);  
   break;  
   }  
   delete this.constraintsByName[name];  
   return this;  
   },
* [¶](#49x2ik5)  
  Update a constraint (Remove + re-add)  
   updateConstraint: function (name, parameters, priority) {  
   return this.removeConstraint(name)  
   .addConstraint(name, parameters, priority);  
   },

# [¶](#2p2csry) Internals

* [¶](#147n2zr)  
  Internal only. Bind constraints from config + options + DOM  
   \_bindConstraints: function () {  
   var constraints = [];  
   var constraintsByName = {};
* [¶](#3o7alnk)  
  clean all existing DOM constraints to only keep javascript user constraints  
   for (var i = 0; i < this.constraints.length; i++)  
   if (false === this.constraints[i].isDomConstraint) {  
   constraints.push(this.constraints[i]);  
   constraintsByName[this.constraints[i].name] = this.constraints[i];  
   }  
    
   this.constraints = constraints;  
   this.constraintsByName = constraintsByName;
* [¶](#23ckvvd)  
  then re-add Parsley DOM-API constraints  
   for (var name in this.options)  
   this.addConstraint(name, this.options[name], undefined, true);
* [¶](#ihv636)  
  finally, bind special HTML5 constraints  
   return this.\_bindHtml5Constraints();  
   },
* [¶](#32hioqz)  
  Internal only. Bind specific HTML5 constraints to be HTML5 compliant  
   \_bindHtml5Constraints: function () {
* [¶](#1hmsyys)  
  html5 required  
   if (null !== this.element.getAttribute('required'))  
   this.addConstraint('required', true, undefined, true);
* [¶](#41mghml)  
  html5 pattern  
   if (null !== this.element.getAttribute('pattern'))  
   this.addConstraint('pattern', this.element.getAttribute('pattern'), undefined, true);
* [¶](#2grqrue)  
  range  
   let min = this.element.getAttribute('min');  
   let max = this.element.getAttribute('max');  
   if (null !== min && null !== max)  
   this.addConstraint('range', [min, max], undefined, true);
* [¶](#vx1227)  
  HTML5 min  
   else if (null !== min)  
   this.addConstraint('min', min, undefined, true);
* [¶](#3fwokq0)  
  HTML5 max  
   else if (null !== max)  
   this.addConstraint('max', max, undefined, true);
* [¶](#1v1yuxt)  
  length  
   if (null !== this.element.getAttribute('minlength') && null !== this.element.getAttribute('maxlength'))  
   this.addConstraint('length', [this.element.getAttribute('minlength'), this.element.getAttribute('maxlength')], undefined, true);
* [¶](#4f1mdlm)  
  HTML5 minlength  
   else if (null !== this.element.getAttribute('minlength'))  
   this.addConstraint('minlength', this.element.getAttribute('minlength'), undefined, true);
* [¶](#2u6wntf)  
  HTML5 maxlength  
   else if (null !== this.element.getAttribute('maxlength'))  
   this.addConstraint('maxlength', this.element.getAttribute('maxlength'), undefined, true);
* [¶](#19c6y18)  
  html5 types  
   var type = Utils.getType(this.element);
* [¶](#3tbugp1)  
  Small special case here for HTML5 number: integer validator if step attribute is undefined or an integer value, number otherwise  
   if ('number' === type) {  
   return this.addConstraint('type', ['number', {  
   step: this.element.getAttribute('step') || '1',  
   base: min || this.element.getAttribute('value')  
   }], undefined, true);
* [¶](#28h4qwu)  
  Regular other HTML5 supported types  
   } else if (/^(email|url|range|date)$/i.test(type)) {  
   return this.addConstraint('type', type, undefined, true);  
   }  
   return this;  
   },
* [¶](#nmf14n)  
  Internal only. Field is required if have required constraint without false value  
   \_isRequired: function () {  
   if ('undefined' === typeof this.constraintsByName.required)  
   return false;  
    
   return false !== this.constraintsByName.required.requirements;  
   },
* [¶](#37m2jsg)  
  Internal only. Shortcut to trigger an event  
   \_trigger: function (eventName) {  
   return this.trigger('field:' + eventName);  
   },
* [¶](#1mrcu09)  
  Internal only Handles whitespace in a value Use data-parsley-whitespace="squish" to auto squish input value Use data-parsley-whitespace="trim" to auto trim input value  
   \_handleWhitespace: function (value) {  
   if (true === this.options.trimValue)  
   Utils.warnOnce('data-parsley-trim-value="true" is deprecated, please use data-parsley-whitespace="trim"');  
    
   if ('squish' === this.options.whitespace)  
   value = value.replace(/\s{2,}/g, ' ');  
    
   if (('trim' === this.options.whitespace) || ('squish' === this.options.whitespace) || (true === this.options.trimValue))  
   value = Utils.trimString(value);  
    
   return value;  
   },  
    
   \_isDateInput: function() {  
   var c = this.constraintsByName.type;  
   return c && c.requirements === 'date';  
   },
* [¶](#46r0co2)  
  Internal only. Returns the constraints, grouped by descending priority. The result is thus an array of arrays of constraints.  
   \_getGroupedConstraints: function () {  
   if (false === this.options.priorityEnabled)  
   return [this.constraints];  
    
   var groupedConstraints = [];  
   var index = {};
* [¶](#2lwamvv)  
  Create array unique of priorities  
   for (var i = 0; i < this.constraints.length; i++) {  
   var p = this.constraints[i].priority;  
   if (!index[p])  
   groupedConstraints.push(index[p] = []);  
   index[p].push(this.constraints[i]);  
   }
* [¶](#111kx3o)  
  Sort them by priority DESC  
   groupedConstraints.sort(function (a, b) { return b[0].priority - a[0].priority; });  
    
   return groupedConstraints;  
   }  
    
  };  
    
  export default Field;