* 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)

# multiple.js

* [¶](#gjdgxs)  
  import $ from 'jquery';  
  import Utils from './utils';  
    
  var Multiple = function () {  
   this.\_\_class\_\_ = 'FieldMultiple';  
  };  
    
  Multiple.prototype = {
* [¶](#30j0zll)  
  Add new $element sibling for multiple field  
   addElement: function ($element) {  
   this.$elements.push($element);  
    
   return this;  
   },
* [¶](#1fob9te)  
  See Field.\_refreshConstraints()  
   \_refreshConstraints: function () {  
   var fieldConstraints;  
    
   this.constraints = [];
* [¶](#3znysh7)  
  Select multiple special treatment  
   if (this.element.nodeName === 'SELECT') {  
   this.actualizeOptions().\_bindConstraints();  
    
   return this;  
   }
* [¶](#2et92p0)  
  Gather all constraints for each input in the multiple group  
   for (var i = 0; i < this.$elements.length; i++) {
* [¶](#tyjcwt)  
  Check if element have not been dynamically removed since last binding  
   if (!$('html').has(this.$elements[i]).length) {  
   this.$elements.splice(i, 1);  
   continue;  
   }  
    
   fieldConstraints = this.$elements[i].data('FieldMultiple').\_refreshConstraints().constraints;  
    
   for (var j = 0; j < fieldConstraints.length; j++)  
   this.addConstraint(fieldConstraints[j].name, fieldConstraints[j].requirements, fieldConstraints[j].priority, fieldConstraints[j].isDomConstraint);  
   }  
    
   return this;  
   },
* [¶](#3dy6vkm)  
  See Field.getValue()  
   getValue: function () {
* [¶](#1t3h5sf)  
  Value could be overriden in DOM  
   if ('function' === typeof this.options.value)  
   return this.options.value(this);  
   else if ('undefined' !== typeof this.options.value)  
   return this.options.value;
* [¶](#4d34og8)  
  Radio input case  
   if (this.element.nodeName === 'INPUT') {  
   var type = Utils.getType(this.element);  
   if (type === 'radio')  
   return this.\_findRelated().filter(':checked').val() || '';
* [¶](#2s8eyo1)  
  checkbox input case  
   if (type === 'checkbox') {  
   var values = [];  
    
   this.\_findRelated().filter(':checked').each(function () {  
   values.push($(this).val());  
   });  
    
   return values;  
   }  
   }
* [¶](#17dp8vu)  
  Select multiple case  
   if (this.element.nodeName === 'SELECT' && null === this.$element.val())  
   return [];
* [¶](#3rdcrjn)  
  Default case that should never happen  
   return this.$element.val();  
   },  
    
   \_init: function () {  
   this.$elements = [this.$element];  
    
   return this;  
   }  
  };  
    
  export default Multiple;