* 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)

# form.js

* [¶](#gjdgxs)  
  import $ from 'jquery';  
  import Base from './base';  
  import Utils from './utils';  
    
  var Form = function (element, domOptions, options) {  
   this.\_\_class\_\_ = 'Form';  
    
   this.element = element;  
   this.$element = $(element);  
   this.domOptions = domOptions;  
   this.options = options;  
   this.parent = window.Parsley;  
    
   this.fields = [];  
   this.validationResult = null;  
  };  
    
  var statusMapping = {pending: null, resolved: true, rejected: false};  
    
  Form.prototype = {  
   onSubmitValidate: function (event) {
* [¶](#30j0zll)  
  This is a Parsley generated submit event, do not validate, do not prevent, simply exit and keep normal behavior  
   if (true === event.parsley)  
   return;
* [¶](#1fob9te)  
  If we didn’t come here through a submit button, use the first one in the form  
   var submitSource = this.\_submitSource || this.$element.find(Utils.\_SubmitSelector)[0];  
   this.\_submitSource = null;  
   this.$element.find('.parsley-synthetic-submit-button').prop('disabled', true);  
   if (submitSource && null !== submitSource.getAttribute('formnovalidate'))  
   return;  
    
   window.Parsley.\_remoteCache = {};  
    
   var promise = this.whenValidate({event});  
    
   if ('resolved' === promise.state() && false !== this.\_trigger('submit')) {
* [¶](#3znysh7)  
  All good, let event go through. We make this distinction because browsers differ in their handling of submit being called from inside a submit event [#1047]  
   } else {
* [¶](#2et92p0)  
  Rejected or pending: cancel this submit  
   event.stopImmediatePropagation();  
   event.preventDefault();  
   if ('pending' === promise.state())  
   promise.done(() => { this.\_submit(submitSource); });  
   }  
   },  
    
   onSubmitButton: function(event) {  
   this.\_submitSource = event.currentTarget;  
   },
* [¶](#tyjcwt)  
  internal \_submit submits the form, this time without going through the validations. Care must be taken to “fake” the actual submit button being clicked.  
   \_submit: function (submitSource) {  
   if (false === this.\_trigger('submit'))  
   return;
* [¶](#3dy6vkm)  
  Add submit button’s data  
   if (submitSource) {  
   var $synthetic = this.$element.find('.parsley-synthetic-submit-button').prop('disabled', false);  
   if (0 === $synthetic.length)  
   $synthetic = $('<input class="parsley-synthetic-submit-button" type="hidden">').appendTo(this.$element);  
   $synthetic.attr({  
   name: submitSource.getAttribute('name'),  
   value: submitSource.getAttribute('value')  
   });  
   }  
    
   this.$element.trigger(Object.assign($.Event('submit'), {parsley: true}));  
   },
* [¶](#1t3h5sf)  
  Performs validation on fields while triggering events. @returns true if all validations succeeds, false if a failure is immediately detected, or null if dependant on a promise. Consider using whenValidate instead.  
   validate: function (options) {  
   if (arguments.length >= 1 && !$.isPlainObject(options)) {  
   Utils.warnOnce('Calling validate on a parsley form without passing arguments as an object is deprecated.');  
   var [group, force, event] = arguments;  
   options = {group, force, event};  
   }  
   return statusMapping[ this.whenValidate(options).state() ];  
   },  
    
   whenValidate: function ({group, force, event} = {}) {  
   this.submitEvent = event;  
   if (event) {  
   this.submitEvent = Object.assign({}, event, {preventDefault: () => {  
   Utils.warnOnce("Using `this.submitEvent.preventDefault()` is deprecated; instead, call `this.validationResult = false`");  
   this.validationResult = false;  
   }});  
   }  
   this.validationResult = true;
* [¶](#4d34og8)  
  fire validate event to eventually modify things before every validation  
   this.\_trigger('validate');
* [¶](#2s8eyo1)  
  Refresh form DOM options and form’s fields that could have changed  
   this.\_refreshFields();  
    
   var promises = this.\_withoutReactualizingFormOptions(() => {  
   return $.map(this.fields, field => field.whenValidate({force, group}));  
   });  
    
   return Utils.all(promises)  
   .done( () => { this.\_trigger('success'); })  
   .fail( () => {  
   this.validationResult = false;  
   this.focus();  
   this.\_trigger('error');  
   })  
   .always(() => { this.\_trigger('validated'); })  
   .pipe(...this.\_pipeAccordingToValidationResult());  
   },
* [¶](#17dp8vu)  
  Iterate over refreshed fields, and stop on first failure. Returns true if all fields are valid, false if a failure is detected or null if the result depends on an unresolved promise. Prefer using whenValid instead.  
   isValid: function (options) {  
   if (arguments.length >= 1 && !$.isPlainObject(options)) {  
   Utils.warnOnce('Calling isValid on a parsley form without passing arguments as an object is deprecated.');  
   var [group, force] = arguments;  
   options = {group, force};  
   }  
   return statusMapping[ this.whenValid(options).state() ];  
   },
* [¶](#3rdcrjn)  
  Iterate over refreshed fields and validate them. Returns a promise. A validation that immediately fails will interrupt the validations.  
   whenValid: function ({group, force} = {}) {  
   this.\_refreshFields();  
    
   var promises = this.\_withoutReactualizingFormOptions(() => {  
   return $.map(this.fields, field => field.whenValid({group, force}));  
   });  
   return Utils.all(promises);  
   },  
    
   refresh: function() {  
   this.\_refreshFields();  
   return this;  
   },
* [¶](#26in1rg)  
  Reset UI  
   reset: function () {
* [¶](#lnxbz9)  
  Form case: emit a reset event for each field  
   for (var i = 0; i < this.fields.length; i++)  
   this.fields[i].reset();  
    
   this.\_trigger('reset');  
   },
* [¶](#35nkun2)  
  Destroy Parsley instance (+ UI)  
   destroy: function () {
* [¶](#1ksv4uv)  
  Field case: emit destroy event to clean UI and then destroy stored instance  
   this.\_destroyUI();
* [¶](#44sinio)  
  Form case: destroy all its fields and then destroy stored instance  
   for (var i = 0; i < this.fields.length; i++)  
   this.fields[i].destroy();  
    
   this.$element.removeData('Parsley');  
   this.\_trigger('destroy');  
   },  
    
   \_refreshFields: function () {  
   return this.actualizeOptions().\_bindFields();  
   },  
    
   \_bindFields: function () {  
   var oldFields = this.fields;  
    
   this.fields = [];  
   this.fieldsMappedById = {};  
    
   this.\_withoutReactualizingFormOptions(() => {  
   this.$element  
   .find(this.options.inputs)  
   .not(this.options.excluded)  
   .each((\_, element) => {  
   var fieldInstance = new window.Parsley.Factory(element, {}, this);
* [¶](#2jxsxqh)  
  Only add valid and not excluded Field and FieldMultiple children  
   if (('Field' === fieldInstance.\_\_class\_\_ || 'FieldMultiple' === fieldInstance.\_\_class\_\_) && (true !== fieldInstance.options.excluded)) {  
   let uniqueId = fieldInstance.\_\_class\_\_ + '-' + fieldInstance.\_\_id\_\_;  
   if ('undefined' === typeof this.fieldsMappedById[uniqueId]) {  
   this.fieldsMappedById[uniqueId] = fieldInstance;  
   this.fields.push(fieldInstance);  
   }  
   }  
   });  
    
   $.each(Utils.difference(oldFields, this.fields), (\_, field) => {  
   field.reset();  
   });  
   });  
   return this;  
   },
* [¶](#z337ya)  
  Internal only. Looping on a form’s fields to do validation or similar will trigger reactualizing options on all of them, which in turn will reactualize the form’s options. To avoid calling actualizeOptions so many times on the form for nothing, \_withoutReactualizingFormOptions temporarily disables the method actualizeOptions on this form while fn is called.  
   \_withoutReactualizingFormOptions: function (fn) {  
   var oldActualizeOptions = this.actualizeOptions;  
   this.actualizeOptions = function () { return this; };  
   var result = fn();  
   this.actualizeOptions = oldActualizeOptions;  
   return result;  
   },
* [¶](#3j2qqm3)  
  Internal only. Shortcut to trigger an event Returns true iff event is not interrupted and default not prevented.  
   \_trigger: function (eventName) {  
   return this.trigger('form:' + eventName);  
   }  
    
  };  
    
  export default Form;