* 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)

# ui.js

* [¶](#gjdgxs)  
  import $ from 'jquery';  
  import Utils from './utils';  
    
  var UI = {};  
    
  var diffResults = function (newResult, oldResult, deep) {  
   var added = [];  
   var kept = [];  
    
   for (var i = 0; i < newResult.length; i++) {  
   var found = false;  
    
   for (var j = 0; j < oldResult.length; j++)  
   if (newResult[i].assert.name === oldResult[j].assert.name) {  
   found = true;  
   break;  
   }  
    
   if (found)  
   kept.push(newResult[i]);  
   else  
   added.push(newResult[i]);  
   }  
    
   return {  
   kept: kept,  
   added: added,  
   removed: !deep ? diffResults(oldResult, newResult, true).added : []  
   };  
  };  
    
  UI.Form = {  
    
   \_actualizeTriggers: function () {  
   this.$element.on('submit.Parsley', evt => { this.onSubmitValidate(evt); });  
   this.$element.on('click.Parsley', Utils.\_SubmitSelector, evt => { this.onSubmitButton(evt); });
* [¶](#30j0zll)  
  UI could be disabled  
   if (false === this.options.uiEnabled)  
   return;  
    
   this.element.setAttribute('novalidate', '');  
   },  
    
   focus: function () {  
   this.\_focusedField = null;  
    
   if (true === this.validationResult || 'none' === this.options.focus)  
   return null;  
    
   for (var i = 0; i < this.fields.length; i++) {  
   var field = this.fields[i];  
   if (true !== field.validationResult && field.validationResult.length > 0 && 'undefined' === typeof field.options.noFocus) {  
   this.\_focusedField = field.$element;  
   if ('first' === this.options.focus)  
   break;  
   }  
   }  
    
   if (null === this.\_focusedField)  
   return null;  
    
   return this.\_focusedField.focus();  
   },  
    
   \_destroyUI: function () {
* [¶](#1fob9te)  
  Reset all event listeners  
   this.$element.off('.Parsley');  
   }  
    
  };  
    
  UI.Field = {  
    
   \_reflowUI: function () {  
   this.\_buildUI();
* [¶](#3znysh7)  
  If this field doesn’t have an active UI don’t bother doing something  
   if (!this.\_ui)  
   return;
* [¶](#2et92p0)  
  Diff between two validation results  
   var diff = diffResults(this.validationResult, this.\_ui.lastValidationResult);
* [¶](#tyjcwt)  
  Then store current validation result for next reflow  
   this.\_ui.lastValidationResult = this.validationResult;
* [¶](#3dy6vkm)  
  Handle valid / invalid / none field class  
   this.\_manageStatusClass();
* [¶](#1t3h5sf)  
  Add, remove, updated errors messages  
   this.\_manageErrorsMessages(diff);
* [¶](#4d34og8)  
  Triggers impl  
   this.\_actualizeTriggers();
* [¶](#2s8eyo1)  
  If field is not valid for the first time, bind keyup trigger to ease UX and quickly inform user  
   if ((diff.kept.length || diff.added.length) && !this.\_failedOnce) {  
   this.\_failedOnce = true;  
   this.\_actualizeTriggers();  
   }  
   },
* [¶](#17dp8vu)  
  Returns an array of field’s error message(s)  
   getErrorsMessages: function () {
* [¶](#3rdcrjn)  
  No error message, field is valid  
   if (true === this.validationResult)  
   return [];  
    
   var messages = [];  
    
   for (var i = 0; i < this.validationResult.length; i++)  
   messages.push(this.validationResult[i].errorMessage ||  
   this.\_getErrorMessage(this.validationResult[i].assert));  
    
   return messages;  
   },
* [¶](#26in1rg)  
  It’s a goal of Parsley that this method is no longer required [#1073]  
   addError: function (name, {message, assert, updateClass = true} = {}) {  
   this.\_buildUI();  
   this.\_addError(name, {message, assert});  
    
   if (updateClass)  
   this.\_errorClass();  
   },
* [¶](#lnxbz9)  
  It’s a goal of Parsley that this method is no longer required [#1073]  
   updateError: function (name, {message, assert, updateClass = true} = {}) {  
   this.\_buildUI();  
   this.\_updateError(name, {message, assert});  
    
   if (updateClass)  
   this.\_errorClass();  
   },
* [¶](#35nkun2)  
  It’s a goal of Parsley that this method is no longer required [#1073]  
   removeError: function (name, {updateClass = true} = {}) {  
   this.\_buildUI();  
   this.\_removeError(name);
* [¶](#1ksv4uv)  
  edge case possible here: remove a standard Parsley error that is still failing in this.validationResult but highly improbable cuz’ manually removing a well Parsley handled error makes no sense.  
   if (updateClass)  
   this.\_manageStatusClass();  
   },  
    
   \_manageStatusClass: function () {  
   if (this.hasConstraints() && this.needsValidation() && true === this.validationResult)  
   this.\_successClass();  
   else if (this.validationResult.length > 0)  
   this.\_errorClass();  
   else  
   this.\_resetClass();  
   },  
    
   \_manageErrorsMessages: function (diff) {  
   if ('undefined' !== typeof this.options.errorsMessagesDisabled)  
   return;
* [¶](#44sinio)  
  Case where we have errorMessage option that configure an unique field error message, regardless failing validators  
   if ('undefined' !== typeof this.options.errorMessage) {  
   if ((diff.added.length || diff.kept.length)) {  
   this.\_insertErrorWrapper();  
    
   if (0 === this.\_ui.$errorsWrapper.find('.parsley-custom-error-message').length)  
   this.\_ui.$errorsWrapper  
   .append(  
   $(this.options.errorTemplate)  
   .addClass('parsley-custom-error-message')  
   );  
    
   return this.\_ui.$errorsWrapper  
   .addClass('filled')  
   .find('.parsley-custom-error-message')  
   .html(this.options.errorMessage);  
   }  
    
   return this.\_ui.$errorsWrapper  
   .removeClass('filled')  
   .find('.parsley-custom-error-message')  
   .remove();  
   }
* [¶](#2jxsxqh)  
  Show, hide, update failing constraints messages  
   for (var i = 0; i < diff.removed.length; i++)  
   this.\_removeError(diff.removed[i].assert.name);  
    
   for (i = 0; i < diff.added.length; i++)  
   this.\_addError(diff.added[i].assert.name, {message: diff.added[i].errorMessage, assert: diff.added[i].assert});  
    
   for (i = 0; i < diff.kept.length; i++)  
   this.\_updateError(diff.kept[i].assert.name, {message: diff.kept[i].errorMessage, assert: diff.kept[i].assert});  
   },  
    
    
   \_addError: function (name, {message, assert}) {  
   this.\_insertErrorWrapper();  
   this.\_ui.$errorClassHandler  
   .attr('aria-describedby', this.\_ui.errorsWrapperId);  
   this.\_ui.$errorsWrapper  
   .addClass('filled')  
   .append(  
   $(this.options.errorTemplate)  
   .addClass('parsley-' + name)  
   .html(message || this.\_getErrorMessage(assert))  
   );  
   },  
    
   \_updateError: function (name, {message, assert}) {  
   this.\_ui.$errorsWrapper  
   .addClass('filled')  
   .find('.parsley-' + name)  
   .html(message || this.\_getErrorMessage(assert));  
   },  
    
   \_removeError: function (name) {  
   this.\_ui.$errorClassHandler  
   .removeAttr('aria-describedby');  
   this.\_ui.$errorsWrapper  
   .removeClass('filled')  
   .find('.parsley-' + name)  
   .remove();  
   },  
    
   \_getErrorMessage: function (constraint) {  
   var customConstraintErrorMessage = constraint.name + 'Message';  
    
   if ('undefined' !== typeof this.options[customConstraintErrorMessage])  
   return window.Parsley.formatMessage(this.options[customConstraintErrorMessage], constraint.requirements);  
    
   return window.Parsley.getErrorMessage(constraint);  
   },  
    
   \_buildUI: function () {
* [¶](#z337ya)  
  UI could be already built or disabled  
   if (this.\_ui || false === this.options.uiEnabled)  
   return;  
    
   var \_ui = {};
* [¶](#3j2qqm3)  
  Give field its Parsley id in DOM  
   this.element.setAttribute(this.options.namespace + 'id', this.\_\_id\_\_);  
    
   /\*\* Generate important UI elements and store them in this \*\*/
* [¶](#1y810tw)  
  $errorClassHandler is the $element that woul have parsley-error and parsley-success classes  
   \_ui.$errorClassHandler = this.\_manageClassHandler();
* [¶](#4i7ojhp)  
  $errorsWrapper is a div that would contain the various field errors, it will be appended into $errorsContainer  
   \_ui.errorsWrapperId = 'parsley-id-' + (this.options.multiple ? 'multiple-' + this.options.multiple : this.\_\_id\_\_);  
   \_ui.$errorsWrapper = $(this.options.errorsWrapper).attr('id', \_ui.errorsWrapperId);
* [¶](#2xcytpi)  
  ValidationResult UI storage to detect what have changed bwt two validations, and update DOM accordingly  
   \_ui.lastValidationResult = [];  
   \_ui.validationInformationVisible = false;
* [¶](#1ci93xb)  
  Store it in this for later  
   this.\_ui = \_ui;  
   },
* [¶](#3whwml4)  
  Determine which element will have parsley-error and parsley-success classes  
   \_manageClassHandler: function () {
* [¶](#2bn6wsx)  
  Class handled could also be determined by function given in Parsley options  
   if ('string' === typeof this.options.classHandler && $(this.options.classHandler).length)  
   return $(this.options.classHandler);
* [¶](#qsh70q)  
  Class handled could also be determined by function given in Parsley options  
   var $handlerFunction = this.options.classHandler;
* [¶](#3as4poj)  
  It might also be the function name of a global function  
   if ('string' === typeof this.options.classHandler && 'function' === typeof window[this.options.classHandler])  
   $handlerFunction = window[this.options.classHandler];  
    
   if ('function' === typeof $handlerFunction) {  
   var $handler = $handlerFunction.call(this, this);
* [¶](#1pxezwc)  
  If this function returned a valid existing DOM element, go for it  
   if ('undefined' !== typeof $handler && $handler.length)  
   return $handler;  
   } else if ('object' === typeof $handlerFunction && $handlerFunction instanceof jQuery && $handlerFunction.length) {  
   return $handlerFunction;  
   } else if ($handlerFunction) {  
   Utils.warn('The class handler `' + $handlerFunction + '` does not exist in DOM nor as a global JS function');  
   }  
    
   return this.\_inputHolder();  
   },  
    
   \_inputHolder: function() {
* [¶](#49x2ik5)  
  if simple element (input, texatrea, select…) it will perfectly host the classes and precede the error container  
   if (!this.options.multiple || this.element.nodeName === 'SELECT')  
   return this.$element;
* [¶](#2p2csry)  
  But if multiple element (radio, checkbox), that would be their parent  
   return this.$element.parent();  
   },  
    
   \_insertErrorWrapper: function () {  
   var $errorsContainer = this.options.errorsContainer;
* [¶](#147n2zr)  
  Nothing to do if already inserted  
   if (0 !== this.\_ui.$errorsWrapper.parent().length)  
   return this.\_ui.$errorsWrapper.parent();  
    
   if ('string' === typeof $errorsContainer) {  
   if ($($errorsContainer).length)  
   return $($errorsContainer).append(this.\_ui.$errorsWrapper);  
   else if ('function' === typeof window[$errorsContainer])  
   $errorsContainer = window[$errorsContainer];  
   else  
   Utils.warn('The errors container `' + $errorsContainer + '` does not exist in DOM nor as a global JS function');  
   }  
    
   if ('function' === typeof $errorsContainer)  
   $errorsContainer = $errorsContainer.call(this, this);  
    
   if ('object' === typeof $errorsContainer && $errorsContainer.length)  
   return $errorsContainer.append(this.\_ui.$errorsWrapper);  
    
   return this.\_inputHolder().after(this.\_ui.$errorsWrapper);  
   },  
    
   \_actualizeTriggers: function () {  
   var $toBind = this.\_findRelated();  
   var trigger;
* [¶](#3o7alnk)  
  Remove Parsley events already bound on this field  
   $toBind.off('.Parsley');  
   if (this.\_failedOnce)  
   $toBind.on(Utils.namespaceEvents(this.options.triggerAfterFailure, 'Parsley'), () => {  
   this.\_validateIfNeeded();  
   });  
   else if (trigger = Utils.namespaceEvents(this.options.trigger, 'Parsley')) {  
   $toBind.on(trigger, event => {  
   this.\_validateIfNeeded(event);  
   });  
   }  
   },  
    
   \_validateIfNeeded: function (event) {
* [¶](#23ckvvd)  
  For keyup, keypress, keydown, input… events that could be a little bit obstrusive do not validate if val length < min threshold on first validation. Once field have been validated once and info about success or failure have been displayed, always validate with this trigger to reflect every yalidation change.  
   if (event && /key|input/.test(event.type))  
   if (!(this.\_ui && this.\_ui.validationInformationVisible) && this.getValue().length <= this.options.validationThreshold)  
   return;  
    
   if (this.options.debounce) {  
   window.clearTimeout(this.\_debounced);  
   this.\_debounced = window.setTimeout(() => this.validate(), this.options.debounce);  
   } else  
   this.validate();  
   },  
    
   \_resetUI: function () {
* [¶](#ihv636)  
  Reset all event listeners  
   this.\_failedOnce = false;  
   this.\_actualizeTriggers();
* [¶](#32hioqz)  
  Nothing to do if UI never initialized for this field  
   if ('undefined' === typeof this.\_ui)  
   return;
* [¶](#1hmsyys)  
  Reset all errors’ li  
   this.\_ui.$errorsWrapper  
   .removeClass('filled')  
   .children()  
   .remove();
* [¶](#41mghml)  
  Reset validation class  
   this.\_resetClass();
* [¶](#2grqrue)  
  Reset validation flags and last validation result  
   this.\_ui.lastValidationResult = [];  
   this.\_ui.validationInformationVisible = false;  
   },  
    
   \_destroyUI: function () {  
   this.\_resetUI();  
    
   if ('undefined' !== typeof this.\_ui)  
   this.\_ui.$errorsWrapper.remove();  
    
   delete this.\_ui;  
   },  
    
   \_successClass: function () {  
   this.\_ui.validationInformationVisible = true;  
   this.\_ui.$errorClassHandler.removeClass(this.options.errorClass).addClass(this.options.successClass);  
   },  
   \_errorClass: function () {  
   this.\_ui.validationInformationVisible = true;  
   this.\_ui.$errorClassHandler.removeClass(this.options.successClass).addClass(this.options.errorClass);  
   },  
   \_resetClass: function () {  
   this.\_ui.$errorClassHandler.removeClass(this.options.successClass).removeClass(this.options.errorClass);  
   }  
  };  
    
  export default UI;