**validator.js**

validator.js is a validation tool that can be used to validate input elements, both as part of a form, or as a single input. Because the validator runs on data attributes and classes, no programming is required for basic usage. validator.js also supports custom validation rules and successful validation function via callbacks, as well as many other options and features.

When an input fails validation, an error message is displayed to alert the user to the problem. Standard error message divs are displayed as a fixed position element so it will not affect the layout of the dom when placed.

**Initialization:**

To initialize the validator all that is required is to call the init function: validator().init(). Ideally this is done in your parent view after the document first loads. Because the validator listens to the input and click events at the document level, it does not need to be initialized again until the user switches web pages. Switching partial views does not require re-initialization.

When initialized, the validator listens for click events for form validation, and input, focus, and blur events for individual input validation and help text. It will also listen for keypress events if you want to restrict the type of characters allowed in an input box. If you want to have an individual input validated on another event besides the input event, to get the validator to listen for that event, simply pass in an array of the events to the init function. For instance, to make the validator listen for the change event, the initialize function would be called as: validator().init([“change”]);



**Basic usage:**

validator.js runs off of data attributes and classes, so no programming is required unless you want to have a custom validation rule. Validating both the form and input level require many of the same attributes and classes, but there are a few distinctions.

**Form validation:**

To use the basic form validation, you do not actually need a form surrounding the input elements; any common parent element like a div or fieldset will work just as well. The parent element needs only one class to be validated: “formValidate”. Although there are additional options that can be supplied (which will be covered later), this is the only class necessary for basic usage. Putting the validate class on the parent element allows the user to remove the class from one or all forms on the page and effectively turn off validation dynamically since the validator won’t work on any form that doesn’t have this class.



The action button on the form will need a data-attribute called “data-form”. The value should be set to the id of the parent div that has the “formValidate” class. This tells the validator which parent element contains all the inputs to be validated, and allows multiple forms on one view to be validated independently.



Any inputs inside the parent element that should be validated against the validator’s internal library of validation rules needs a “data-type” attribute. This value should be a comma delimited list of the validation rules to test against that input. The data-type attribute uses a library of validation rules to test the value of the input. As an example, if you have an email input, then the attribute should be set as “data-type= email’”. If the email input is also a required field, add an attribute called “data-requiredfield” and the validator will check for input too.



When validating an input, if it is a required field, and it fails the required validation, no other rules will be tested against that particular input. Only if the field passes the required validation or if it isn’t a required field will the remaining rules be tested.

If you don’t want to use the library of validation rules or have a need to write your own rule, you can do that with the “data-customrules” attribute which is covered later.

Finally, for the form to take action when validated, an attribute called “data-formaction” is used on the parent element. The value should be set to a string name of your success function. For instance, if you want to make an AJAX call to the controller with the contents of the form on successful validation, and you named that AJAX function “formSubmit()”, then the data-formaction attribute should be set to “formSubmit” – no open/close parentheses. After validation occurs, if all input elements passed validation, then the formSubmit() function is called.



**Input Validation:**

Validating a single input is similar to validating a form, except all attributes and classes are put on the input itself and it is tested against the input event by default unless you pass in another event for the validator to listen for. In this case, the input requires a class called “inputValidate”. As in form validation, if you want to turn validation off for an input, simply remove the inputValidate class and it will not be tested.

The input also requires the same data-type attribute as a form input requires and it is used in the same way. Supply a comma delimited list of the validation rules to test against, and on each input event for that element, it will be tested against whichever rules were specified. If multiple validation rules are specified, beyond required, and the input fails several of them, the error messages for each failed rule will be displayed.



**Additional Options/Features:**

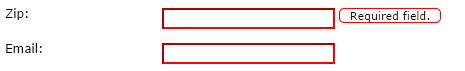
**Grouping Errors:**

**Hover**

By default, all error messages will be displayed all the time next to the inputs that failed validation. There are two options available to change the way the error message are displayed.

To make the error message only display when hovering over a specific input, add a class to the parent element or the individual input called “hover”. This will hide the error message div when not hovering over an input that failed validation.





**Group by input**

You can also group all the errors in a statically placed div next to each input rather than have fixed positioned divs that “stack” out from the input in whatever direction is specified. To group errors by input, add a class to the parent container for form validation or the input for single input validation called “groupByInput”.



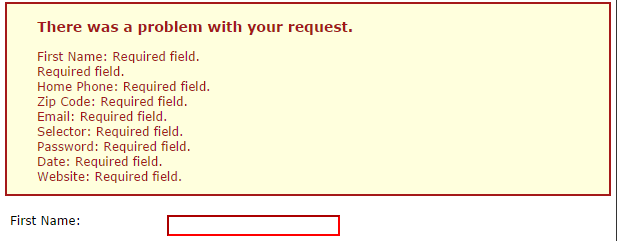


**Group by form**

If you would like to group all the errors in a div rather than display them next to each input, you need to make a container element for the error messages. You can place the container anywhere on the page and style it however you like. The parent form needs an added class called “data-grouperrors” with the value set to the id of the error container. By default, only the error messages are moved into the group container, so to indicate which error belongs to which input, each input should have an attribute called “data-errorprefix”, the value of which will the prepended to the error message and can be used to tie each error message to the input it’s referring to.





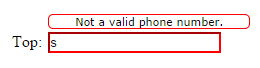


**Error message placement:**

**Placing error relative to input**

By default, the form will display all error message divs to the right of the input that failed validation. If you want to change the placement of the error div, add an attribute called “data-location” to the input. The values can be: top, bottom, left, or right. The error message div is still placed in relation to the input, but it will on the corresponding side of that input. For left and right placements, if there is more than one validation rule that is failed, the error messages will be placed further out to the left or right respectively. For top and bottom, the errors will be stacked on top of each other in the appropriate direction.



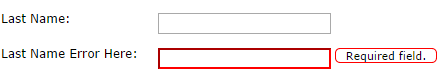


**Display the error message on another element**

Occasionally, when using another library’s input widgets, two inputs will be used even though only one is visible at any given time. Effectively one input overlays the other depending on the focus of the inputs. If the input you’re validating is hidden when the user blurs, then the error message div won’t be displayed if the hover option is turned on, and any css you use to highlight the input field also won’t be seen. To avoid this, an attribute called “data-displayon” can be added to an input. This will allow validation to occur for the input, but it will display on the visible input that is used when blurred. The value should be a css class name (with the preceding “.”), or an id (with the preceding “#”), as well as the location of the other element in the dom relative to the validated input; ie up or down. Because the dynamically added inputs used by widgets don’t usually have an id, the class name will work, but the validator needs to know to search above or below the input that it is currently validating. It will use the first match in the dom that it finds in the direction it is searching. If no match is found, it will display the error message div on the actual input it validated.

This functionality can also be used to change the location of the error message div even if there isn’t an overlayed element on top of it. If for some reason you want to display the error message somewhere else, just add the displayon attribute and give it the element and location (up or down) of the element it should display the error message on, and the validator will adjust the location accordingly.

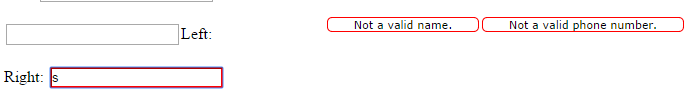




**Offsetting error messages**

By default, the error message div is placed 5 pixels away from whatever element it is displayed on. To adjust the offset, add an attribute called “data-offset” to the input. The value should be the number of pixels you want to offset the div from the element where the error message div is being displayed. All subsequent error message divs for that input will continue to be displayed with that offset.





**Miscellaneous:**

**Help text**

If you would like help text to display for your input when a user focuses on it, add a span above the input in the dom, but as close to a sibling as possible to it. Give the span a class called “helpText” and “hideMessage”, and whatever text is inside of the span will be displayed next to the input when the user focuses and removed when they blur the input. This will also be affected by the data-location attribute, data-displayon attribute, and the data-offsetwidth and data-offsetheight attributes of the input. So if error messages are displayed on the bottom of the input with a width offset of 20 pixels, the help text will be as well. Note that the span should be hidden from view when the page loads, so adding the “hideMessage” class is necessary in addition to the “helpText” class. This needs to be done by the user because while the validator could hide the spans on the page when it loads, any dynamically added spans, including those from different views, will not be hidden since the validator is only initialized once.





**Scroll Listener**

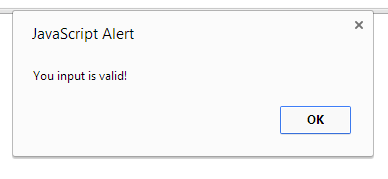
If the form or input is displayed inside of a scrollable container like a modal or popup, an attribute called “data-modalid” can be added so that the validator listens to the scroll event of the modal in addition to window. The value should be set to the id of the popup container. This will ensure that the error messages’ locations are moved along with both the window and modal when scrolled. This will also stop displaying the error message div when the input is scrolled “above” or “below” the popup in the case of fixed position divs.



**Input success function**

Like the form level validation, an input can have a success function called when it is successfully validated. Because the validator listens to the input event by default when validating a single input, the success function should be used with caution as the input could go in and out of a valid state while typing. To add a success function to an input give it an attribute called “data-inputaction” with the value being the string name of the function the validator should call; the same way as the formaction attribute.





**Removing error messages programmatically**

By default, the validator will remove all error messages each time an input is validated, and only display the errors for which the input failed validation. However, you may want to remove validation errors when a user action takes place that should reset the validation or makes the validation moot; like when closing a modal that has failed validation. In this case, call the validator function “removeErrors”. If no parameter is passed in, the validator will remove all error messages from the body of the document. Otherwise, you can pass in the id of the form or input element you want the validator to remove the errors from.



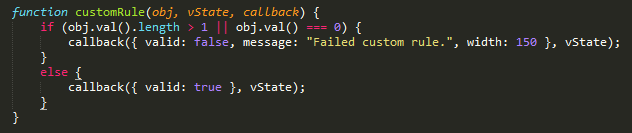
**Custom validation rules and callback functions:**

**Custom validation rules**

To add custom validation rules to an input, use the “data-customrules” attribute. This works the same way as the data-type attribute; it take a comma delimited list of custom validation function names, without the open/close parens, and tests the input against them. When calling the custom rule functions, three parameters will be passed to the function. The first is the input currently being tested wrapped in a JQuery object. It does not need to be used or returned, but if your custom function is generic enough to be used on multiple inputs, then you can use the object without having to make a custom rule for each input. The second parameter is an object that represents the current state of validation for that input. Nothing needs or should be done with this parameter save returning it through the callback function. The last parameter is the callback function.

After your custom rule tests the input, use the callback function to return to the validator. The callback function takes two parameters. The first parameter should be an object specifying if the input passed the validation rule, and if not what the error message to be displayed is. It can optionally take in a width parameter if you want to change it from its default value. If the input passed the custom validation rule, no message or width is required. The second parameter is a validation state for the input that was passed into your custom rule. See the image below for the structure and fields of the validation object and how to use the callback function.



****

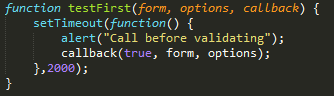
**Call before validation function**

The form can have a function that is called before validation. This is done in much the same way that the data-formaction works. Add an attribute called “data-beforevalidate” and set the value to the string name of the function you want to be called without the open/close parens. If there is any work that needs to be done before the validator is run, it can be done in this function. When the before validate function is done running, callback with a value of true to continue validation or false and validation will not be run on the form. This can be used as another way to dynamically turn off validation, but unlike removing the “validate” class, this can be non-deterministic as it can done at runtime after the action button has been clicked.

The call before validation functionality works just like the custom rule functionality. Like the custom rules, the call before validation function will take in three parameters. The first is the form element currently being validated wrapped in a JQuery object. This parameter is for your use should you need the form object inside the function. The second parameter is the options that have been set on your form for validation and need to be left untouched and returned to the validator in the callback function. The last parameter is the callback function and should be used to return to the validator.

The callback function takes three parameters when you’re ready to return to the validator. The first is a boolean used to indicate if the validation should proceed. The second and third parameters are the form element and options that were passed to your call before function.





**CSS classes:**

The validator applies classes to the various elements it uses so you can apply custom styling to these objects. All inputs that have failed one or more validation rules are given a class of “invalid”. Use this to style the input with a red border or any other visual indicator of failed validation. The error message divs are given a class called “errorMessage”, and when grouped, the error message container is given a class called “groupedErrors”.