**CommonValidation**

1. **INTENT**

To have a common validation framework throughout site with features like:

* Centrally or Selectively(App wise) controllable
* Highly customizable
* Easily extendible (beyond in-built functionalities)
* Minimum integration effort at user’s end
* NUI Based Plug & Play script

1. **STRUCTURE**

* **Prerequisites:** “NUI Framework” and “commonValidation” javascript files.
* **Code Structure Divisions:** 
  + HTML *(To structure the form fields and the error message placehoders)*
  + Plugin Call *(To initialize common validator for the required form(s) on the html page)*
  + Error-Code List *(For mapping error codes against the error messages and for extended functionalities)*

1. **USAGE**

**3.1 HTML**

1. The form that needs to be validated using commonValidator, **must have** its **“name”** attribute specified.
2. The **“name” and “id”** attributes of all the form fields are **not mandatory**.
3. To show **inline error messages**, there must be an element within the form which can be an html tag like <div>,<span>,<p>,<i>,<b> etc. having its “id” attribute set as the “id” or “name” of the form field with which it is related, appended with **“\_err”** and must have a class “**erLbl**” specified on it.

***Example:***

<input type="text" **id="reqI"** name="reqN" rel="required:1001"/>

<i **class="erLbl" id="reqI\_err"**></i>

OR

<input type="text" id="reqI" **name="reqN"** rel="required:1001"/>

<i **class="erLbl"**  **id="reqN\_err"**></i>

1. For help texts, specify the required help text in the **“placeholder”** attribute of the particular form element.

*NOTE: The behavior of the help text will slightly vary in the browsers that does not support the “placeholder” attribute (i.e. IE-6,7,8,9).*

1. To associate various form fields to the validation types and their corresponding error messages/functionalities, there must be a **“rel”** attribute specified on that form field. The **“rel”** attribute specifies the combination(s) of the validation types that needs to be applied on the particular field and the corresponding error code separated by a “:”(colon)(comma separated if multiple) as one part and the event(s)(comma separated if multiple) to initiate validation as second part separated by a “|”(pipe). The various ways of specification are as bellow:
2. **Simple :** If default event(s)(“defaultEvents” parameter) is specified in the plugin call.

***Example:***

<input type="text" id="reqI" name="reqN" **rel="required:1001"**/>

OR

<input type="text" id="reqI" name="reqN" **rel="required:1001,alpha:1002"**/>

1. **Events in “rel” :** If default event(s)(“defaultEvents” parameter) is not specified or some custom events are required on a field

***Example:***

<input type="text" id="reqI" name="reqN" **rel="required:1001|blur"**/>

OR

<input type="text" id="reqI" name="reqN" **rel="required:1001,alpha:1002|blur,keyup"**/>

1. **Character Range :** To set character range validation on a form field, set attributes “minlength” or “minL” for minimum value and “maxlength” or “maxL” for maximum value, on the form element.

***Example:***

<input type="text" id="rngI" name="rngN" **minlength="5" maxlength="10"** rel="charRange:1001|keyup,blur"/>

OR

<input type="text" id="rngI" name="rngN" **minL="5" maxL="10"** rel="charRange:1001|keyup,blur"/>

1. **Value Range :** To set value range validation on a form field, set attributes “minval” or “minV” for minimum value and “maxval” or “maxV” for maximum value, on the form element.

***Example:***

<input type="text" id="rngI" name="rngN" **minval="10" maxval="100"** rel="valRange:1001|keyup,blur"/>

OR

<input type="text" id="rngI" name="rngN" **minV="10" maxV="100"** rel="valRange:1001|keyup,blur"/>

1. **Scope of Range Validations(For Character Range and Value Range) :** To specify whether to validate the for the range within the specified values or outside the specified values, set attribute “scope” on the form field with value “in” to validate for the range within the specified values or “out” to validate for the range outside the specified values. If not specified, the scope is set to “in” by default.

***Example:***

<input type="text" id="rngI" name="rngN" minlength="5" maxlength="10" **scope="in"** rel="charRange:1001|keyup,blur"/>

OR

<input type="text" id="rngI" name="rngN" minval="10" maxval="100" **scope="out"** rel="valRange:1001|keyup,blur"/>

1. **“rel” for Submit Button :** To associate a custom function with the submit button, specify the corresponding error code and the event, separated by a “|” pipe.

***Example:***

<input id="sBtn" type="button" value="Submit" **rel="3001|click"**/>

**3.2 PLUGIN CALL**

1. The call to initialize validation on a particular form or forms is:

**commonValidator.validate({ “*parameters”* });**

1. The **“*parameters”***  plugin are:
2. **formNames :** Accepts the form names on which commonValidator needs to be initialized. Form name can be a string for a single form or an array of form names for multiple forms. At least one form name is mandatory.
3. **errors :** Accepts the name of the error message array. If not mentioned, the default name ***“commonErrList”*** of the array is set.
4. **styles :** Accepts three different types of classes to be implemented on three different types of error scenarios. Namely, “errorClass” for error message, “softMandClass” for soft mandatory error messages and “okClass” for no error fields. If any or all of these are not mentioned, then classes “err”, “softMand” and “ok” are set as default classes for “errorClass”, “softMandClass” and “okClass” parameters respectively.
5. **clearOnFocus(Boolean) :** Accepts “True” or “False”. If enabled, it clears the error message and error class from the field on focus. If not specified, its is set as “False” by default.
6. **messageBox :** Set to show a separate message box in case of errors in the form. Accepts parameters as bellow:
7. **id :** Accepts the “id” of the message box. Mandatory.
8. **content :** To specify the type of content to be shown inside the message box. If not set, the message box will be shown with static html content written in it. It accepts three sub parameters mentioned as below:

* **customContent :** Accepts custom message string to be shown inside the message box.
* **errorCount :** Set “true” to show the number of errors with a default message or set a custom message string including tag “[errCount]” anywhere in the message string where the Error Count needs to be shown.
* **errorMessages(Boolean) :** Set “true” to show all the error messages related to all the invalid fields of the form. Is not specified, it is set to “false” by default.

1. **hideOthers(Boolean) :** Set “true” to hide the message boxes associated to other forms. If not specified, it is set to “false” by default.
2. **inlineErrors(Boolean) :** Set “true” or “false” to show or hide the inline error messages related to each form field. If not specified, set to “true” by default.
3. **defaultEvents :** Accepts event names as a string(if single event) or array(if multiple events) to be attached with all the form fields for validation trigger.
4. **submitButton :** If the submit button is not of the type “submit” then set this parameter with the “id” of the submit button as a string.

***Example Code for Plugin Call :***

commonValidator.validate({

formNames : 'myFormName',

errors : commonErrList,

styles : {errorClass:'err',okClass:'ok',softMandClass:'softMand'},

clearOnFocus : false,

messageBox : {id:'msgBoxID', content: {customContent:'My custom content for the

MsgBox', errorCount:'Total [errCount] errors found - Custom.',

errorMessages:true}, hideOthers:false},

inlineErrors : true,

defaultEvents : 'blur',

submitButton : 'buttonID'

});

**3.3 ERROR CODE LIST**

It is an associative array which maps error codes against the error messages or various other cases/functionalities explained as under.

1. **Simple :** For simply mapping the error messages against a unique error codes as key:value pairs.

***Example :***

var commonErrList = {

1001 : 'Required Field can not be left blank'

}

NOTE: Error message defined in this way will be shown inside the error placeholder defined with id as the id of its corresponding form field appended with “\_err”.

1. **Custom ID :** For populating the error message inside an error placeholder having id different from the default(mentioned as above) id, the error code will be mapped against an associative array with keys “msg” and “id”. The error message string will be set against the “msg” key and the id of the error placeholder will be set against the “id” key.

***Example:***

var commonErrList = {

1001 : {msg : 'Required Field can not be left blank' , id : 'customID'}

}

NOTE: Error message defined in this way will be shown inside the error placeholder defined with id “customID”.

1. **Current Value :** For displaying the value entered in the field to be validated inside the error message, use tag “[currVal]” within the error message string.

***Example:***

var commonErrList = {

1001 : '[currVal]" is not valid. Please enter a valid value'

}

1. **Character Range :** In character range field, for displaying the values of minimum and maximum characters allowed, use tags “[MinL]” and “[MaxL]” respectively within the error message string.

***Example:***

var commonErrList = {

1001 : 'Minimum [MinL] and Maximum [MaxL] characters are allowed'

}

1. **Value Range :** In value range field, for displaying the minimum and maximum values allowed, use tags “[MinV]” and “[MinV]” respectively within the error message string.

***Example:***

var commonErrList = {1001 : 'The value can not be less than [MinV] and greater than [MaxV]'}

1. **Custom Regular Expression :** For specifying a custom regular expression(overwriting the default regular expression), the error code will be mapped against an associative array with keys “msg” to set the error message string and key “regEx” to set the custom regular expression.

***Example:***

var commonErrList = {1001 : '{msg : 'Only alphabets are allowed' , regEx : /^[a-zA-Z]+$/}'}

1. **Custom Function :** For specifying a custom validation criteria, the error code will be mapped against a function defined by then user, specifying the logic of the custom function and returning “error message string” in case of error and “false” in case of no error.

***Example:***

***1: Writing the function inline***

var commonErrList = {

1001 : function(){

if($n('#someID').val()<10){

return 'Value can not be less than 10'

}

else{return false}

}

}

***2: Passing the function as an object***

var myFunc = function(){

if($n('# someID ').val()<10){

return 'Value can not be less than 10'

}

else{return false}

}

var commonErrList = {1001 : myFunc}

***3: Accessing the form field object inside the custom function***

var myFunc = function (obj){

if(obj.val()<10){

return 'Value can not be less than 10';

}

else{return false}

}

var commonErrList = {1001 : myFunc}

***4: Accessing in-built validations inside the custom function***

In-built validations, syntax of the call is:

**commonValidator.checkValids(*validation-type*,*form-element*)**

where **“*validation-type”*** is one of the validation type keywords(same as used in “rel” attributes i.e. “required”, “alpha”, “num” etc.) to specify the type of validation and **“*form-element*”** is the object of the element on which this validation type needs to be applied.

***Example:***

var myFunc = function (obj){

var rtn=false;

if(commonValidator.checkValids('num',obj)){

rtn='"[currVal]" is not a numeric value'

}

return {msg:rtn,id:'someID'}

}

var commonErrList = {1001 : myFunc}

NOTE: Please refer to section-3.4 for the list of all the in-built validation types.

* 1. **In-Built validation types and their corresponding function names**

|  |  |
| --- | --- |
| **Keyword** (for “rel”) | **Description** |
| **required** | Mandatory fields |
| **softReq** | Soft mandatory fields |
| **alphaDS** | Allows alphabets with dot and spaces |
| **alpha** | Allows alphabets only |
| **num** | Allows numeric only |
| **float** | Allows floating point values |
| **alphanum** | Allows alphanumeric values |
| **email** | Validates email address formats |
| **specialChar** | Does not allow special characters |
| **charRange** | Allows number of characters according to the specified range |
| **valRange** | Allows values according to the specified range |
| **checked** | Validates for checking checkboxes and radio buttons |
| **selected** | Validate select boxes |
| **custom** | To specify a custom functions for validating form fields |
| **noValidate** | To submit the form without validating |

* 1. **Applying validation on Text Editor:**

In text editor scenario, the “textarea” element used for the text editor creation should have “rel” attribute with “custom” keyword and code combination(as only custom validation functions are applicable in case of text editor) and another attribute “editor” with its value set to the object name of the editor associated to it as a string.

***Example:***

1. **Editor Call :**

**$editor**=new editorWindow();

$editor.createEditor("css/editor.css", "", "mailBody", "$editor", referMail", "sub", "nxt");

1. **Textarea element for editor:**

<textarea name="myTextArea" id="mailBody" **editor="$editor" rel="custom:3001"** > </textarea>

* 1. **For fields with show/hide functionality**

1. **Show Logic :**

The code block where the show logic is specified, an additional function “commonValidator.showElement” needs to be called with the id(if single) or array of the ids(if multiple) of the fields to be shown/hidden mentioned as parameter of the function

***Example:***

$n('#showElements).addEvent('click',function(){

… show logic …

**commonValidator.showElement(['id1','id2']);**

});

1. **Hide Logic :**

The code block where the hide logic is specified, an additional function “commonValidator.hideElement” needs to be called with the id(if single) or array of the ids(if multiple) of the fields to be shown/hidden mentioned as parameter of the function

***Example:***

$n('#hideElements).addEvent('click',function(){

… hide logic …

**commonValidator.hideElement([id1,'id2']);**

});

* 1. **Getting validate status of the form if submit button is not of the type “submit” :**

If submit button is not of the type “submit”, then to get the validate status of form on clicking the submit button, there is a function “commonValidator.isValid(*‘id’*)” where *‘id’* can be the id a form or an individual form element whose validation status needs to be checked. The function returns “true” or “false”. If there is no error in the form or the individual field then value returned is “false” else the value is “true”.

***Example:***

$n('#someButton').addEvent('click',function(){

if(commonValidator.isValid('formID')){alert('There is no error')}

else{alert('There is some error')}

});

* 1. **Submitting the form without validating it :**

To submit the form on the click of a button but without validating it. Specify the button with “type” other than “submit” and specify “rel” attribute of the button as “noValidate” and in the plugin call, specify the id of the button in the “submitButton” parameter.

***Example:***

**HTML:**

<input id="someID" type="button" value="Submit" rel="noValidate"/>

**Plugin call :**

commonValidator.validate({

…Other Parameters… ,

submitButton : ‘someID’

});