Software Design Documentation

## Document Outline

* Introduction
* System Overview
* Design Considerations
  + Assumptions & Dependencies
  + General Constraints
  + Goals & Guidelines
  + Development Method
* Design Overview
* Detailed System Design

## Introduction

The Software Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Software Design Document are narrative and graphical documentation of the software design for the project

#### Purpose

The purpose of the Software Design Document is to provide a description of the design of a system fully enough to allow for software development to proceed with an understanding of what is to be built and how it is expected to built. The Software Design Document provides information.

#### Scope

This Software Design Document is for a base level system which will work as a proof of concept for the use of building a system the provides a base level of functionality to show feasibility for large scale production use.

## 

## System Overview

formValidator.js is a jQuery plugin for validating forms. formValidator.js will watch all forms with an inputs with a data attribute 'data-formvalidate'. Only inputs with said attribute is validated by the plugin. Other inputs in the form are left untouched. Users will be able to add their own validation rules. Custom rules are shared among all formValidator instances. Configure individual formValidator instances by passing a configuration object. The plugin will generate error messages and css classes.

## Design Considerations

#### Assumptions & Dependencies

* This plugin uses html5 attributes and ES5 JavaScript language constructs. This plugin is compatible with only modern browser that support above features.
* jQuery library is a dependency for this plugin.

#### General Constraints

* For using async patterns, the plugin needs to be run on servers (local/online) because of browser security restrictions.
* Ajax calls are subjected to browser security restrictions and hence asyncURL values cannot reference cross-site resources.
* This plugin only supports text like input elements. It will not support form elements like select, radio, checkbox etc.

#### Goals & Guidelines

* The design places emphasis on automating the validation with very little configuration on the user’s part for getting started. However, it is flexible and configurable enough for advanced use cases.

#### Development Methods

formValidator.js uses established jQuery plugin development best practices. It uses a prototypal inheritance model to represent the forms. Uses a Pubsub model for events.

## Design Overview

#### Initializing

The plugin automatically initializes all forms that having inputs with a data attribute 'data-formvalidate'. Only inputs with said attribute is validated by the plugin. Other inputs in the form are left untouched.

Plugin initialised from HTML

<input type="text" data-formvalidate="number">

The plugin can also be initialised from javascript

$('form').formValidate();

#### Setting Validation type

By default the plugin supports 3 validation types. ‘number’, ‘required’, ‘email’. Multiple rules can be added to the same input using a comma separated string.

<input type="text" data-formvalidate="required, email">

#### 

#### Configurations

Configure individual formValidator instances by passing a configuration object.

**attributeUsed** (String)

Used to set a different attribute to track validation other than data-formvalidate.

**triggerUsed** (String)

any jQuery input related event to trigger validation (ex: keyup, blur).

**scroll** (Boolean)

scroll to the first invalid input when form is submitted.

**focusFirstField** (Boolean)

focus first invalid input when form is submitted.

**asyncURL** (URI String)

The server URL for fetching the pattern from the server. (ex: <http://example.com/pattern>)

**asyncPattern** (Boolean)

Whether to use async pattern fetched from asyncURL for validation.

Ex:

$('form').formValidate({

attributeUsed: 'data-formvalidate',

triggerUsed: 'blur',

scroll: true,

focusFirstField: true,

asyncURL: '',

asyncPattern: true

});

#### 

#### 

#### Adding Custom rules:

addRules() method is used to add user defined rules by passing custom configuration object. Custom rules are shared among all formValidator instances.

**name** (lowercase string without spaces )

name to be used in data-formvalidate attribute.

**errorMessage** (String)

error triggered when state is invalid. This will be displayed as error message.

**pattern** (RegExp)

Any javascript compatible RegExp pattern to validate against the input value.

**trim** (Boolean)

Setting to determine if leading and trailing space be removed when processing the input value.

$.formValidate.addRules({

name: 'integer',

errorMessage: 'Entry must be an Integer',

pattern: /^[+-]?\d$/i,

trim: false

});

#### Error message & CSS classes

To display error messages, both the form and input need the 'name' attribute. Error messages will be updated to any element that has attribute 'data-formmsg'. To display form level errors use the form's name as the value of data-formmsg. Input level errors, data-formmsg shoul contail a value of *'form name'* and *'input name'* concatenated with a dot (\*);

<form name="myForm">

<input type="text" data-formvalidate="number" name="age">

<span data-formmsg="myForm.age">Input's errors will be displayed here.</span>

</form>

<div data-formmsg="myForm">

Form level errors will be displayed here.

</div>

formValidate.js doesn't add any CSS rules. Instead, it adds classes *'valid'* and *'invalid'* to input elements when validation is triggered. It also adds classes related to validation types, ex: 'required' type will trigger 'required-invalid' & 'required-valid' classes for corresponding states. Form element has *'valid'* and *'invalid'* classes updated on submit event.

## Detailed System Design

Class **FormValidate**

**Methods:**

**init** (elem, options)

Arguments:

elem : (DOM). Form DOM element passed for the plugin

options: (Object). User’s configuration object for overriding default values.

Desc: init() acts as a constructor function. It initializes class specific properties and call bindEvents() to bind the events to the inputs.

**bindEvents** ()

Desc: This function binds the user defined event to the inputs having the selector attribute which will trigger the validations.

**validateInput** ($input)

Arguments:

$input: (jQuery wrapped input element). The passed input will be validated by the function

Returns

isAllTestsPassed (Boolean). Return true if the input passes all assigned validations.

Desc: This function will iterate through all the validation types the passed input possess and validate it with the internal list of validation rules. This function manages calling the methods responsible for displaying the error messages and updating the css classes.

**validateAllInputs**()

Desc: This method is called on submit and iterates through all inputs (having the selector criteria) and call validateInput($input) on them. Once all inputs are validated, it calls asynchronous method if it is enabled.

**validateAsync**()

Arguments:

url (URI string): The URL for checking asynchronous pattern.

subject (URLencode String): The value to be posted for checking.

Returns

Deferred Object with validation status

Desc: This function accepts the value and uses a POST request to the URL provided.

**onAsyncValidationSucces** (isAsyncTestPassed, isAllInlineTestsPassed)

Arguments:

isAsyncTestPassed (Boolean): status of async validation.

isAllInlineTestsPassed (Boolean): status of inline validations

Desc:This method is called when the async validation is success. This will call methods that will set the status and error class for form element and async input to reflect success.

**onAsyncValidationFailure** (isAsyncTestPassed, isAllInlineTestsPassed)

Arguments:

isAsyncTestPassed (Boolean): status of async validation.

isAllInlineTestsPassed (Boolean): status of inline validations.

Desc:This method is called when the async validation is success. This will call methods that will set the status and error class for form element and async input to reflect failure

**toggleInputValidationClass** ($input, test, isInputValid, isAllTestsPassed)

Arguments:

$input (jQuery input DOM element)

test (String): The validation type.

isInputValid (Boolean): current validation type status .

isAllTestsPassed (Boolean): validation status for all assigned validation types.

Desc: This method will toggle ‘valid’ and ‘invalid’ class for supplied input.

**toggleFormValidationClass** (isFormValid)

Arguments:

isFormValid (Boolean): form validation status

Desc: This method will toggle ‘valid’ and ‘invalid’ class for the form.

**updateErrorMessages** ($input, inputErrorList)

Arguments:

$input (jQuery input DOM element): input tested for calidation.

inputErrorList (Array): String array of error messages

Desc: Generates error messages for form-level and inline elements based on the form and input names.

**scrollToElement** ($element)

Argument:

$element (jQuery DOM)

Desc: Scrolls the page to the specified DOM element.

**setFocusToInput** ($input)

Argument:

$input (jQuery input DOM element)

Desc: focuses on the specified DOM element.