Stingray bootstrap API ver. 3.2.1.1

Core files

PLEASE DO NOT MODIFY ANY OF THE LISTED FILES!

*New mobile first (bootstrap) client:*

* js/stingray-min.js : main client module.
* css/stingray-min.css : main client styles.
* js/libraries-min.js : 3rd party libraries.
* jQuery 1.12.4
* jQuery UI 1.12.1
* Sizzle Engine 2.2.1
* Bootstrap 3.3.5
* Form Validation 0.7.0
* Bootbox.js 4.4.0
* Mousetrap 1.5.3
* Sticky Plugin 1.0.4
* Super Signature 1.5.0.2
* Bootstrap Notify 3.1.3
* Auto Numeric 1.9.45
* css/libraries-min.css : 3rd party styles.
* Glyphicons
* Font Awesome
* Bootstrap
* jQuery UI
* FormValidation
* Animate
* js/debug-min.js : development tools.
* css/debug-min.css : development tools styles.
* vrmEditor.html : standalone editor or editor in its own tab.
* debug-\*.html : development tools.
* Log viewer
* Reqlist viewer
* Watchpoint viewer

*Classic absolute positioned client:*

* script/main.js : main module with client and editor.
* script/require.js : third party libraries used by absolute positioned client and editor.
* style/global.css : styles used by 3rd party libraries.

*Other files required by Stingray system and its editor:*

* bootstrap.vrm : starting template to create new bootstrap page.
* absolute.vrm : starting template to create new classic page.
* empty.vrm : starting template to create new non WYSIWYG HTML page.
* stingrayrules\0\admintabs.vrm : server logic used by web editor.
* stingrayrules\0\logoff.vrm : logout logic for classic Stingray systems.
* stingrayrules\0\logout.vrm : logout logic for bootstrap Stingray systems.
* include\ui-default.inc : shared HTML to be injected into bootstrap pages.
* include\scriptfunctions.inc : list of server scripting functions used in editor and help.

Landing page

Main landing html document must have references to the following core files:

*In the header <HEAD>:*

<link href="css/libraries-min.css" type="text/css" rel="stylesheet">

<link href="css/stingray-min.css" type="text/css" rel="stylesheet">

*At the end of body <BODY>:*

<script type="text/javascript" src="js/libraries-min.js"></script>

<script type="text/javascript" src="js/stingray-min.js"></script>

The main landing page must have a div container with required id and vrmName attributes where pages will be delivered.

<div id="bootstrapContent" vrmName="firstVrmToSubmitThisPage">

Page submit

Buttons submitting current page and navigating to another can have onclick with new Post call. An example of such button with required attributes in red:

<button type="button" class="btn btn-default" ref="button" name="goNext" value="nextBtn" onclick="MP.Comm.Post(this)" default="true" validate="true">Next</button>

* ref : required reference for editor.
* onclick : usually MP.Comm.Post or Get to serialize and submit the page.
* name : will add SubmitAct variable into ReqList which can be used to identify the clicked button (an element) and navigate server side (post/preprocess) script.
* value : will add SubmitVal variable into ReqList.
* default : a Boolean allowing submitting page or its section using the enter key.
* validate : a Boolean making page validation to occur.

Default button(s)

Pressing an enter key inside a field can trigger the page’s main default button or a selected default button if the page has more than one.

Older framework does not support multiple default buttons on a single page. This framework does with several techniques to accomplish desired behavior:

1. There is a single default button on the page with default attribute set to true [default="true"]. This is the default behavior of the older framework.
2. Add an empty default attribute [default=""] to any element to indicate a section or group with its own default button to trigger on enter key.
3. Add a default attribute [default="#myDefaultBtn"] to any element to indicate a section or group with value set to jQuery selector indicating a specific button to trigger on enter key. This button may or may not have to be part of same section. Also the button does not need default attribute nor its value set to true in order to be triggered.

Components

As you can see above, every editor component must have attribute ref. Without it, the editor will not recognize the component and will not correctly load it. Allowed ref (reference) attributes are:

container, row, column, div, input, checkbox, radio, textarea, select, button, label, panel, widget, and js.

Each component may have an optional init event allowing to customize it during loading. Its value can be any JS code or function with any number of parameters declared in CustomScript, GlobalScript or any other place. An example of such event could be:

ref=”widget” init=”Widgets.rollUp(this)”

Widget

Widgets can hold any HTML and execute any related JS code to render and attach needed functionality. For a reusable widget we suggest to have globally accessible widgets.css and widgets.js in the system. An average widget component may have the following properties (required in red):

* HTML – holds the HTML structure
* class – for optional styling
* init – optionally can initialize, update or attach interactive behavior

Note that HTML property shown in editor is not a true property when editing raw code or out of editor. For fully working examples see widgets in included examples.zip

Maximum Processing Core - MP object

The new client javascripts are housed in a single MP object which contains all core functionality. One of the most used objects is communication which can be accessed as MP.Comm.

Initialize(config);

* config : a configuration object to change or customize behavior of the entire core including all its sub components.

ShowVersion(message);

* message : an optional message to display with version of the core.

Communication – MP.Comm object

Initialize(config);

* config : a configuration object to change or customize communication behavior. Default options:

sessionTimeOut: 3600

sessionWarningTime: 120

InProgress();

* returns a boolean identifying if there is a communication in progress.

ShowProgress();

* displays an overlay with spinner. Returns a reference to the spinner.

HideProgress(blurInput);

* hides the overlay with spinner. Returns a reference to the spinner.
* blureInput : blures input if provided.

LogOut (message);

* securely logs user out with or without confirmation dialog.
* message: optional message to display in the confirmation dialog.

OpenWindow (url, tabName, config);

* url: opens this URL in new tab or window.
* tabName: tab or window name to load page into.
* config: optional configuration for new tab or window.
* returns instance of opened tab or window.

Serialize(html);

* Serialize input’s name and values into URL encoded string which can be used directly in URLs and post data.
* html: required HTML string, jQuery selector or jQuery object.
* returns URL encoded string of serialized inputs.

SessionOK();

* returns true for correct (not invalidated) session.

Post(config);

config can be either plain ‘this’ to pass an element invoking the event, ‘URL’ to pass a link of VRM page, jQuery object to mimic an element invoking the event or a configuration object:

* url : If provided Post will be the same as old CustomRequest. If set it can be a string with correct URL to a VRM with or without properly encoded parameters.
* Returns an object with the entire post sent to the server.

Get(config);

config can be either plain ‘URL’ to pass a link of VRM page, ‘this’ or jQuery object to pass the element invoking the event, or a configuration object:

* url : required string with correct URL to a VRM with or without properly encoded parameters.
* Returns an object with the entire request sent to the server.

Optional parameters of configuration (config) object:

* targetElement : specifies which part of the page will be updated with server response. If not provided target will be determined from VRM script. Providing it will override the target set in VRM script. It can be either element, jQuery selector string, jQuery object, or null. Setting it to null will not update anything even if set in VRM script!
* serializeElement : an optional element to specify which part of the page to serialize and add to the request or post data. If not provided the entire page will be serialized. It can be either element, jQuery selector string, jQuery object, or null. Setting it to null will not serialize anything!
* eventElement : an optional element to add its name as SubmitAct and/or value as SubmitVal into the RL. If not provided the element triggering the event will be used instead. It can be either element, jQuery selector string, jQuery object, or null. Setting it to null will not add anything into the RL!
* data : an optional string, JSON or jQuery object to serialize and add to request’s URL or post.
* noProgress : false // to enable or disable page spinner overlay during communications.
* noValidationDialog : false // to show or hide dialog with validation errors.
* validate : false // forces page validation to occur even for buttons with no validate attribute set.
* done : function(data) {} // custom callback function triggered on success.
* fail : function(error, data) {} // custom callback function triggered on failure.
* cache : false
* timeout : 180000 // timeout in milliseconds.
* async : true // highly not suggested to make synchronous calls!

Examples:

The familiar old code:

Communication.SerialRequest( $(“#rightColumn”), false, this );

Can now be coded as simple as:

MP.Comm.Post(this);

Or as complex as:

MP.Comm.Post({

serializeElement : "#myDivToSerialize",

targetElement : "#updateContentOfThisDiv",

eventElement : "#notReallyClickedButton",

data : {

message: "feel like adding this to RL",

reason: "to show how cool this Post is"

},

timeout : 60000,

noProgress : true,

noValidationDialog : true,

validate : true,

done : function(html) {

$('#updateDiv').html(html);

},

fail : function(error, data) {

MP.Utils.ShowErrorMessage(error);

}

});

Same for old LinkRequest:

Communication.LinkRequest("myVrm.max?Name=#SName#");

Can now be coded as:

MP.Comm.Get ("myVrm.max?Name=#SName#");

Or with all the bells and whistles as cousin Post.

Note that Session (ID), unique URL identifier (T), and language preference (CSUL) are added automatically and do not need to be added.

Core Events

CustomScript.OnLoad(container)

A page specific event executed after each response triggered by Get/Post communication methods.

GlobalScript.OnLoad(container)

A system wide event executed after each response triggered by Get/Post communication methods.

* Containers are the updated target element allowing its further manipulation.

OnSerializedValue(value, name, element)

Allows value manipulation prior submitting to the server. If present in either custom or global script the event MUST return a value. Event in local custom script have precedence over global script.

* Value – input’s value which can be manipulated and returned for submit.
* Name – input’s name. Only for reference.
* element – jQuery reference to the input.

OnValidationErrors(errors)

Triggers when page validation result errors. If present in either custom or global script and returns true it will prevent a core validation error dialog to show. Event in local custom script have precedence over global script.

* errors – an array with validation errors.

Dialogs – MP.Dialog object

Initialize(config);

* config : a configuration object to change or customize behavior of dialogs. Default options:

customDialog: window.bootbox,

defaults: {

size: 'small',

backdrop: true,

closeButton: false

}

Alert(message, callback);

Confirm(message, callback);

Prompt(message, callback, value);

Custom(message, callback);

Error(message, callback);

HideAll();

Required parameters are in red, optional in black. All methods return reference to the displayed dialog. Required message attribute of custom dialogs can be either plain string or a full configuration object:

* message : required message/dialog body. It can be either jQuery selector string or jQuery object.
* title : optional dialog title.
* callback : triggered on close. Required for Confirm and Prompt dialogs which will pass back a result parameter.
* onEscape : event allowing the user to dismiss the dialog by hitting ESC.
* value : an initial value for Prompt dialog.
* buttons : an object describing bottom buttons (see below).
* size : can be ‘large’ or ‘small’
* animate : allows dialog animation.
* closeButton : whether the dialog should have a close button or not.
* backdrop : whether the dialog should be modal or not.
* Returns reference to the shown dialog.

An example of simple button object:

okBtn: {

label: 'OK',

className: 'btn-primary',

callback: function() {custom code}

}

*For full API go to* [*http://bootboxjs.com/documentation.html*](http://bootboxjs.com/documentation.html)

Notifications – MP.Notify object

Initialize(config);

* config : a configuration object to change or customize behavior of dialogs. Default options:

customNotify: $.notify

defaults: {}

CloseAll(location);

* location : optional filter such as class to target only specific notifications.

Info(message, config);

Success(message, config);

Warning(message, config);

Danger(message, config);

Custom(message, config);

Required parameters are in red, optional in black. All methods return reference to the displayed notification.

Required message attribute can be either plain string or an object with options:

icon: 'glyphicon glyphicon-warning-sign',

title: 'Title',

message: 'Notification',

url: 'https://github.com/mouse0270/bootstrap-notify',

target: '\_blank'

Optional config attribute is a configuration object with options:

element: 'body',

position: null,

type: "info",

allow\_dismiss: true,

newest\_on\_top: false,

showProgressbar: false,

placement: {

from: "top",

align: "right"

},

offset: 20,

spacing: 10,

z\_index: 1031,

delay: 5000,

timer: 1000,

url\_target: '\_blank',

mouse\_over: null,

animate: {

enter: 'animated fadeInDown',

exit: 'animated fadeOutUp'

},

onShow: null,

onShown: null,

onClose: null,

onClosed: null,

icon\_type: 'class',

template: HTML

*For full API go to*: <http://bootstrap-notify.remabledesigns.com/>

Validation – MP.Validator object

Initialize(config);

* config : a configuration object to change or customize behavior of validations. Default options:

framework: 'bootstrap',

trigger: 'blur',

icon: {

valid: '',

invalid: '',

validating: ''

},

requiredClass: 'req-field',

currency: {

aSign: '$'

}

Add (container, config);

Adds validation to a container or entire page (affects inputs with validator attribute)

* container : optional container to add predefined validations to. It can be either an element, jQuery selector string or jQuery object. Not providing it will add validation to the entire page.
* config : a configuration object to change behavior of container’s validation.
* Returns a form validation instance.

Remove (container);

Completely removes validation from a container or entire page. Call Add to re-activate validation.

* container : optional container to remove all validations from. It can be either an element, jQuery selector string or jQuery object. Not providing it will remove validation from the entire page.
* Returns a container from which validation was removed.

Reset(container);

Resets validation state of a container or entire page.

* container : optional container to reset validation on. It can be either an element, jQuery selector string or jQuery object. Not providing it will reset the entire page.
* Returns a form validation instance.

Validate (container, config);

* container : optional container to validate. It can be either an element, jQuery selector string or jQuery object. Not providing it will validate the entire page.
* config : a configuration object to change behavior of this validation.
* Returns an array of found validation errors.

RequiredFields(enable, selector, blankMsg);

Adds or removes required validator from selected elements, container or entire page leaving all other validators as they are.

* enable: a Boolean to enable or disable input to be required.
* selector: optional element, jQuery selector string or jQuery object to apply. Not providing it will set the entire page.
* blankMsg: optional message to show for empty fields.
* Returns a set of affected elements.

EnableFields(enable, selector, validatorName);

Enables or disables validations for selected elements, container or entire page.

* enable: a Boolean to enable or disable input’s validations.
* selector: optional element, jQuery selector string or jQuery object to apply. Not providing it will set the entire page.
* validatorName: optional name to set specific validator only. Blank will set all input’s validators.
* Returns a set of affected elements.

AddFields(selector, config);

Adds validation to the selected elements, container or entire page (affects inputs with validator attribute)

* selector: optional element, jQuery selector string or jQuery object to add validation to. Not providing it will add validation to the entire page.
* config : a configuration object to change behavior of input’s validation.
* Returns a set of affected elements.

RemoveFields(selector);

Removes validation from selected elements, container or entire page. Call AddFields to re-activate validation.

* selector: optional element, jQuery selector string or jQuery object to remove validations from. Not providing it will remove validation from the entire page.
* Returns a set of affected elements.

ResetFields(selector);

Resets validation state of selected elements, container or entire page.

* selector: optional element, jQuery selector string or jQuery object to reset validation of. Not providing it will reset validation of the entire page.
* Returns a set of affected elements.

InProgress();

* returns a boolean identifying if validation is currently in progress.

Correct:

An object with functions to update string values into correct forms. This list will grow overtime.

* date (value)
* phone (value)
* ssn (value)
* fein (value)

Default:

An object with default core validators. This list will grow overtime:

* required (blankMsg)
* select (blankMsg) // required for dropdowns, -1 values are un-selected!
* state (invalidMsg, isRequired, blankMsg)
* zip (invalidMsg, isRequired, blankMsg)
* phone (invalidMsg, isRequired, blankMsg)
* email (invalidMsg, isRequired, blankMsg)
* emailList (invalidMsg, isRequired, blankMsg)
* currency (invalidMsg, isRequired, blankMsg) // $1,235.00
* ssn (invalidMsg, isRequired, blankMsg) // 123-45-6789
* fein (invalidMsg, isRequired, blankMsg) // 12-3456789
* vin (invalidMsg, isRequired, blankMsg)
* driverLicense (state, invalidMsg, isRequired, blankMsg) // e.g. ‘FL’, ‘#input’, ‘select[name=st]’
* creditCard (invalidMsg, isRequired, blankMsg)
* cvv (ccField, invalidMsg, isRequired, blankMsg)
* rtn (invalidMsg, isRequired, blankMsg)
* same (field, invalidMsg, isRequired, blankMsg) // compares itself to another field
* different (fields, invalidMsg, isRequired, blankMsg) // compares itself to another field/s (CSV)
* equals (value, invalidMsg, isRequired, blankMsg) // compares values
* regexp (pattern, modifiers, invalidMsg, isRequired, blankMsg) // e.g. regexp("^[a-z' .]+$", "I")
* time (invalidMsg, isRequired, blankMsg) // 12:35 AM/PM
* date (invalidMsg, isRequired, blankMsg)
* dateAfter (date, invalidMsg, isRequired, blankMsg)
* dateBefore (date, invalidMsg, isRequired, blankMsg)
* dateBetween (date1, date2, invalidMsg, isRequired, blankMsg)
* int (invalidMsg, isRequired, blankMsg)
* intLessThan (int, invalidMsg, isRequired, blankMsg)
* intGreaterThan (int, invalidMsg, isRequired, blankMsg)
* intBetween (int1, int2, invalidMsg, isRequired, blankMsg)
* float (invalidMsg, isRequired, blankMsg)
* floatLessThan (float, invalidMsg, isRequired, blankMsg)
* floatGreaterThan (float, invalidMsg, isRequired, blankMsg)
* floatBetween (float1, float2, invalidMsg, isRequired, blankMsg)

Parameters in red are required.

All default validators can be simply used by providing single validator attribute instead of several data-fv-xxx validation attributes:

validator="required()" or "required('Please enter a middle name')"

validator="select()" or "select('Please choose a gender.')"

validator="emailList()" or "emailList('Agent emails are invalid', true, ‘Please enter least one agent email’)"

*Examples of validators which may auto correct their values:*

* phone

1002003000 -> 100-200-3000

* date, dateBefore, dateAfter, dateBetween

7403 -> 7/4/2003

72403 -> 7/24/2003

742003 -> 7/4/2003

7242003 -> 7/24/2003

07242003 -> 07/24/2003

*An example of custom validation to validate form including hidden inputs:*

var errs = MP.Validator.Validate(drv, {

excluded: [':disabled']

});

if (errs.length)

MP.Dialog.Error(errs.join('<br>'));

*Examples of validation selector parameters:*

':input'

'#myInput'

$('.myClass')

$('#myDiv').find('[isRed=yes]')

*An example of custom validator:*

validator="CustomScript.CheckDOB()"

// public validator function in local custom script

cs.CheckDOB = function() {

    // create new required validator or set it directly to the custom validation object below

    var cfg = MP.Validator.Default.required('Please enter Date of Birth.');

    // add my custom validator to it

    $.extend(cfg.validators, {

        callback: {

           callback: function (value, validator, field) {

                // if blank let the required validator handle it

                if (!value)

                    return true;

                // correct the value by core standards and update the input

                value = MP.Validator.Correct.date(value);

                field.val(value);

                // custom validation logic starts here, DOB no less than 15 years from today

                var val = new Date(value);

                if (isNaN(val.getDate()))

                    return false;

                var now = new Date(),

                    year = now.getFullYear(),

                    min = new Date(now);

                min.setFullYear(year - 15);

                var b = (val < min);

                // display an error dialog only when leaving changed input. Not on page submit!

                if (!b && !MP.Validator.InProgress())

                    MP.Dialog.Error('Driver must be at least 15 years old.');

                // return the validation result

                return b;

            },

            message: 'Driver must be at least 15 years old.'

        }

    });

    // the custom validation configuration MUST be returned back to the core!

    return cfg;

}

*For full list of validators go to* [*http://formvalidation.io/validators/*](http://formvalidation.io/validators/)

*For full API go to* [*http://formvalidation.io/api/*](http://formvalidation.io/api/)

Utilities – MP.Utils object

Initialize(config);

* config : a configuration object to change or customize utilities. Default options:

tooltip:

selector: ".hasTooltip"

defaults:

placement: "top"

trigger: "hover focus"

popover:

selector: ".hasPopover"

defaults:

placement: "top"

trigger: "enter"

datepicker:

selector: ".isDatepicker"

addClass: "select\_date"

defaults:

changeYear: "true"

changeMonth: "true"

yearRange: "1916:2018"

IsObject(variable);

IsFunction (variable);

IsNumber (variable);

IsString (variable);

* variable : whether or not it is of required type.

Trim(string);

RemoveWhiteSpaces(string);

* returns modified value of provided string.

AddCore(container, config);

* container: will add core functionality to a container or element after optional changes to the HTML structure. It can be either an element, jQuery selector string or jQuery object.
* config : optional parameters to update HTML structure of the provided container prior attaching core features:

<input isCss="form-control hasTooltip isDatepicker" isVal="date()" ...>

MP.Utils.AddCore(clonnedDIV, {

renameAttr: {

isCss: 'class',

isVal: 'validator'

}

});

Other options including all default configuration options:

{

renameClass: {

isDP: 'isDatepicker',

hasTP: 'hasTooltip'

},

removeAttr: ['class', 'validator'],

removeClass: ['isDatepicker', 'hasTooltip']

}

MakeGUID();

* returns global unique identifier string.

GetXmlString(xmlDocument);

* returns string of serialized XML document.

SetRadioOfName (name, value);

* updates state and UI of bootstrap decorated radio groups by name.

FirstVersionSameOrAboveSecond(version1, version2);

* compares string versions in ‘x.x.x.x’ formats including 3, 2 or single octet.

ParseURL([url]);

* returns parsed URL object of optional string parameter or browser’s URL.

Autocomplete & AutocompleteList

$.autocompleteList is simplified $.autocomplete similar to our old one. Required arguments in red:

$.autocompleteList({

url: 'vrmName.max',

data: {name: 'value', ...}, // same as in Get/Post

varName: 'customVariableName', // otherwise will be “turn” in RL

done: function(array, status, jqXHR) {

// if used MUST return the array argument in order to show!

return array;

},

fail: function(error, status, jqXHR) {

// return true to suppress the native core error dialog!

}

});

Response set in VRM:

Response\_ContentType = 'list';

Response\_Content = 'line feed separated list of strings';

Note that Session (ID) does not need to be added into data or url.

*For full API of new autocomplete go to:* <http://api.jqueryui.com/autocomplete/>

*Differences:* <http://www.learningjquery.com/2010/06/autocomplete-migration-guide>

Tooltips

To add tooltip to an element ensure it has following class and attribute:

<input class="hasTooltip" title="Tooltip text"/>

Default behavior can be changed by passing new configuration into MP.Utils.Initialize():

tooltip:

selector: ".hasTooltip"

defaults:

placement: "top"

trigger: "hover focus"

*For all default options go to* <http://getbootstrap.com/javascript/#tooltips-options>

Popovers

To add popover to an element ensure it has following class and attribute:

<input class="hasPopover" data-content="Popover text"/>

Default behavior can be changed by passing new configuration into MP.Utils.Initialize():

popover:

selector: ".hasPopover"

defaults:

placement: "top"

trigger: "enter"

*For all default options go to* <http://getbootstrap.com/javascript/#popovers-options>

Date Pickers

To add date picker to an element ensure it has the following class:

<input class="isDatepicker"/>

Default behavior can be changed by passing new configuration into MP.Utils.Initialize():

datepicker:

selector: ".isDatepicker"

addClass: "select\_date"

defaults:

changeYear: "true"

changeMonth: "true"

yearRange: "1916:2018"

*For all default options go to* <http://api.jqueryui.com/datepicker/>

Messaging – MP.WebSocket object

Allows bi-directional messaging between clients and server using browser’s WebSockets. Each client can have open many independent channels addressable by application names. System messages can be displayed as dialogs or notifications. Other application messages must have their own custom handlers.

Initialize(config);

* config : a configuration object to change default setting. Default options:

{

host: '', // Host name if different than loaded webroot

port: 5711,

autoReconnectMS: 30000,

systemApplication: '', // Name to enable system notifications

ssl: false

}

Add(name, config);

* name : a unique name used to identify an application.
* config : a configuration object to customize this application. Available callback handlers:

{

onMessage: function(dataObj, wsObj) {},

onOpen: function(event, wsObj) {},

onClose: function(event, wsObj) {},

onError: function(dataObj, wsObj) {}

}

* returns the new application’s WebSocket object.

Remove(name);

* name : application to remove.

Send(name, data);

* name : application to send data to.
* data: plain string or object with properties to be serialized and sent to the server.
* returns the application’s WebSocket object.

Close(name);

* name : application to close. Blank name will close all existing applications.

Find(name);

* returns application’s WebSocket object by name.

Count();

* returns number of existing applications.

An example of adding a simple application:

MP.WebSocket.Add('workFlow', {

onMessage: function(msg) { MP.Notify.Info(msg.text); },

onClose: function() { MP.Notify.Warning('Workflow is offline!'); }

));