Redactor 8 API & Settings

API

getObject
Returns an object of the Redactor.

```
var obj = $('#redactor').getObject();
```

getEditor
Returns an object of the Redactor's editable layer.

```
var editor = $('#redactor').getEditor();
```

getCode
Returns the Redactor's HTML-code.

```
var html = $('#redactor').getCode();
```

setCode
Puts an HTML code into the Redactor.

```
$('#redactor').setCode('<p>text</p>');
```

getText
Returns Redactor's html as plain text.

```
var text = $('#redactor').getText();
```

getSelected
Returns Redactor's selected html.

```
var selected_html = $('#redactor').getSelected();
```

insertHtml
Puts an HTML code before the cursor.

```
$('#redactor').insertHtml('<p>insert</p>');
```

destroyEditor
Recovers `textarea` by removing `iframe`.

```
$('#redactor').destroyEditor();
```

setFocus
Sets a focus to the Redactor.

```
$('#redactor').setFocus();
```

execCommand
This option allow you to execute the execCommand.

```
$('#redactor').execCommand('bold');
$('#redactor').execCommand('formatblock', '<h5>');
$('#redactor').execCommand('inserthtml', '<p>text</p>');
```

Settings

lang
Default setting is `'en'`.

First you must download your language.

To setup a new language in Redactor you should do two things:

- Connect the language file.
- Call Redactor using the new language option.

For example (or see live example):

```
<script src="redactor/langs/es.js"></script>
<script type="text/javascript">
$(document).ready(
    function()
    {         $('#redactor').redactor({ lang: 'es' });
    }
);
</script>
```

direction
Default setting is `ltr`

Sets text direction (see live example).

```
$('#redactor').redactor({ direction: 'rtl' });
```

toolbar
Default setting is `'object'`

If you want to turn a toolbar off set the toolbar option to `false`:

```
$('#redactor').redactor({ toolbar: false });
```

toolbarExternal
Default setting is `false`

if you want the toolbar is loaded in a separate layer from the editor, call setup (see example):

```
$('#redactor').redactor({
    toolbarExternal: '#your-toolbar-id'
});
```

buttons
By default, this setting contains the following array of toolbar's buttons:

```
['html', '|', 'formatting', '|', 'bold', 'italic', 'deleted', '|',
'unorderedlist', 'orderedlist', 'outdent', 'indent', '|',
'image', 'video', 'file', 'table', 'link', '|',
'fontcolor', 'backcolor', '|', 'alignment', '|', 'horizontalrule']

// additional buttons
// 'underline', 'alignleft', 'aligncenter', 'alignright', 'justify'
```

If you wish to set your own array, set it in this option:

```
$('#redactor').redactor({
    buttons: ['html', '|', 'formatting', '|', 'bold', 'italic']
});
```

buttonsAdd
If you need to add a new button to the toolbar, simply do the following:

```
$('#redactor').redactor({
    buttonsAdd: ['new_button']
});
```

Or with divider:

```
$('#redactor').redactor({
    buttonsAdd: ['|', 'new_button']
});
```

buttonsCustom
This option allows you to add your own buttons with callback to the toolbar.

For more information about creating your own buttons, go to Toolbar section.

```
$('#redactor').redactor({
    buttonsAdd: ['new_button'],
    buttonsCustom: {
        new_button: {
            title: 'New Button',
            callback: function(obj, event, key) { … }
        }
    }
});
```

Or:

```
$('#redactor').redactor({
    buttons: ['html', '|', 'formatting', '|', 'bold', 'italic',
'|', 'new_button'],
    buttonsCustom: {
        new_button: {
            title: 'New Button',
            callback: function(obj, event, key) { … }
        }
    }
});
```

formattingTags
Default setting is `['p', 'blockquote', 'pre', 'h1', 'h2', 'h3', 'h4']`.

Set up formatting drop down list.

```
$('#redactor').redactor({
    formattingTags: ['h1', 'h2']
});
```

source
Default setting is `true`.

Show/hide the HTML source button on the toolbar.

```
$('#redactor').redactor({ source: false });
```

iframe
Default setting is `false`

Option to run the editor iframe with custom styles (see example):

```
$('#redactor').redactor({ iframe: false });
```

css
Default setting is `false`

Load custom styles with iframe (see example):

```
$('#redactor').redactor({
    iframe: true,
    css: 'http://site.com/your_stylesheet_file.css'
});
```

or

```
$('#redactor').redactor({
    iframe: true,
    css: '/styles/your_stylesheet_file.css'
});
```

focus
Default setting is `false`.

This option allows you to set whether Redactor gets cursor focus on load or not.

```
$('#redactor').redactor({ focus: true });
```

shortcuts
Default setting is `true`

Turns on/off keydown / keyup shortcuts functionality.

```
$('#redactor').redactor({ shortcuts: false });
```

autoresize
Default setting is `true`.

This option turns on height autoresizing, which depends on the amount of text inputted into the text layer.

```
$('#redactor').redactor({ autoresize: false });
```

cleanup
Default setting is `true`.

Turns on/off a text's cleanup on paste.

```
$('#redactor').redactor({ cleanup: false });
```

fixed
Default setting is `false`.

If this option is turned on, Redactor's toolbar will remain at the top of the browser's window at all times.

```
$('#redactor').redactor({ fixed: true });
```

fixedTop
Default setting is `0`.

This option setup css top property of a fixed toolbar (in pixels).

```
$('#redactor').redactor({ fixed: true, fixedTop: 50 });
```

fixedBox
Default setting is `false`.

This option makes a fixed toolbar to the width of the editor.

```
$('#redactor').redactor({ fixed: true, fixedBox: true });
```

convertLinks
Default setting is `true`

With this option turned on, Redactor will automatically replace URLs with hyperlinks.

```
$('#redactor').redactor({ convertLinks: false });
```

convertDivs
Default setting is `true`

With this option turned on, Redactor will automatically replace divs to paragraphs.

```
$('#redactor').redactor({ convertDivs: false });
```

autosave
Default setting is `false`

If you set a path to the handler file, Redactor automatically sends the code at default intervals. Autosave will send variable `$_POST['textareaname']`.

```
$('#redactor').redactor({ autosave: '/save.php' });
```

interval
Default setting is `60`

Autosave interval (in seconds).

```
$('#redactor').redactor({
    autosave: '/save.php',
    interval: 30
});
```

autosaveCallback
Default setting is `false`

```
$('#redactor').redactor({
    autosave: '/save.php',
```

```
    autosaveCallback: function(data, redactor_obj) { alert(data); }
});
```

imageGetJson
Default setting is `false`

A path to json file, which stores data about files that have already been uploaded.

```
$('#redactor').redactor({ imageGetJson: '/images.json' });
```
json file example:

```
[
    { "thumb": "/img/1m.jpg", "image": "/img/1.jpg", "title":
"Image 1", "folder": "Folder 1" },
    { "thumb": "/img/2m.jpg", "image": "/img/2.jpg", "title":
"Image 2", "folder": "Folder 1" },
    { "thumb": "/img/3m.jpg", "image": "/img/3.jpg", "title":
"Image 3", "folder": "Folder 1" },
    { "thumb": "/img/4m.jpg", "image": "/img/4.jpg", "title":
"Image 4", "folder": "Folder 2" },
    { "thumb": "/img/5m.jpg", "image": "/img/5.jpg", "title":
"Image 5", "folder": "Folder 2" }
]
```

** title and folder are optional*

imageUpload
Default setting is `false`

This option sets a path to the file that is responsible for uploading images. The handler will receive an array `$_FILES['file']`.

```
$('#redactor').redactor({ imageUpload: '/image_upload.php' });
```
The handler should return JSON, for example:

```
$array = array(
    'filelink' => '/tmp/images/'.$file
);

echo stripslashes(json_encode($array));
```

JSON example:

```
{ "filelink": "/images/img.jpg" }
```

imageUploadCallback
Default setting is `false`

```
$('#redactor').redactor({
    imageUpload: '/image_upload.php',
    imageUploadCallback: function(obj, json) { … }
});
```

imageUploadErrorCallback
Default setting is `false`

Please see example.

```
$('#redactor').redactor({
    imageUpload: '/image_upload.php',
    imageUploadErrorCallback: function(obj, json) { … }
});
```

fileUpload
Default setting is `false`

This option sets a path to the file that is responsible for uploading files. The handler will receive an array `$_FILES['file']`.

```
$('#redactor').redactor({ fileUpload: '/file_upload.php' });
```

The handler should return JSON, for example:

```
$array = array(
    'filelink' => '/files/'.$_FILES['file']['name'],
    'filename' => $_FILES['file']['name']
);

echo stripslashes(json_encode($array));
```

JSON example:

```
{ "filelink": "/files/file.doc", "filename": "Filename" }
```

fileUploadCallback
Default setting is `false`

```
$('#redactor').redactor({
    fileUpload: '/file_upload.php',
    fileUploadCallback: function(obj, json) { … }
});
```

fileUploadErrorCallback
Default setting is `false`

Please see example.

```
$('#redactor').redactor({
    fileUpload: '/file_upload.php',
    fileUploadErrorCallback: function(obj, json) { … }
});
```

uploadCrossDomain
Default setting is `false`

This setting turns on/off drag and drop upload because it does not work for cross-domain upload.

```
$('#redactor').redactor({
    uploadCrossDomain: true
});
```

uploadFields
Default setting is `false`

You can set up an hidden fields for upload (see live example).

The field value for "field1" will be taken from input by ID selector:

```
$('#redactor').redactor({
    uploadFields: {
     'field1': '#field1'
    }
});
```
The field "field2" will automatically generated with value "12345".

```
$('#redactor').redactor({
    uploadFields: {
     'field2': '12345'
    }
});
```
In the upload script you can take them from POST data.

overlay
Default setting is `true`

Turns on and off overlay for modal Redactor's windows.

```
$('#redactor').redactor({ overlay: false });
```

tabindex
Default setting is `false`

```
$('#redactor').redactor({ tabindex: 2 });
```

minHeight
Default setting is `false`

```
$('#redactor').redactor({ minHeight: 200 // pixels });
```

callback
Default setting is `false`

Callback after an editor build.

```
$('#redactor').redactor({
    callback: function(obj) { … }
});
```

keyupCallback
Default setting is `false`

```
$('#redactor').redactor({
    keyupCallback: function(obj, event) { … }
});
```

keydownCallback
Default setting is `false`

```
$('#redactor').redactor({
    keydownCallback: function(obj, event) { … }
});
```

execCommandCallback
Default setting is `false`

This callback calls a function after any changes in Redactor by execCommand.

```
$('#redactor').redactor({
    execCommandCallback: function(obj, command) { … }
});
```

observeImages
Default setting is `true`

Enable or disable by clicking on the image to display a modal window image settings.

```
$('#redactor').redactor({
    observeImages: false
});
```

colors
Default setting is `array`:

```
colors: [
    '#ffffff', '#000000', '#eeece1', '#1f497d', '#4f81bd',
'#c0504d', '#9bbb59', '#8064a2', '#4bacc6', '#f79646', '#ffff00',
    '#f2f2f2', '#7f7f7f', '#ddd9c3', '#c6d9f0', '#dbe5f1',
'#f2dcdb', '#ebf1dd', '#e5e0ec', '#dbeef3', '#fdeada', '#fff2ca',
    '#d8d8d8', '#595959', '#c4bd97', '#8db3e2', '#b8cce4',
'#e5b9b7', '#d7e3bc', '#ccc1d9', '#b7dde8', '#fbd5b5', '#ffe694',
    '#bfbfbf', '#3f3f3f', '#938953', '#548dd4', '#95b3d7',
'#d99694', '#c3d69b', '#b2a2c7', '#b7dde8', '#fac08f', '#f2c314',
    '#a5a5a5', '#262626', '#494429', '#17365d', '#366092',
'#953734', '#76923c', '#5f497a', '#92cddc', '#e36c09', '#c09100',
    '#7f7f7f', '#0c0c0c', '#1d1b10', '#0f243e', '#244061',
'#632423', '#4f6128', '#3f3151', '#31859b', '#974806', '#7f6000']
```

You can call the Redactor with custom colors array, for example:

```
$('#redactor').redactor({
    colors: ['#000000', '#eeece1', '#1f497d', '#4f81bd', '#c0504d',
'#9bbb59', '#8064a2', '#4bacc6', '#f79646', '#ffff00']
```

```
});
```

air
Default setting is `false`

You can turn on an air-mode (see live example):

```
$('#redactor').redactor({
    air: true
});
```

airButtons
Default setting is `array`

```
['formatting', '|', 'bold', 'italic', 'deleted', '|',
'unorderedlist', 'orderedlist', 'outdent', 'indent', '|',
'fontcolor', 'backcolor']
```

You can set up a custom buttons for air-mode:

```
$('#redactor').redactor({
    airButtons: ['formatting', '|', 'bold', 'italic', 'deleted']
});
```

wym
Default setting is `false`

Turns on a visual structure mode (see live example):

```
$('#redactor').redactor({
    wym: true
});
```

allowedTags
Default setting is `array`

```
["code", "span", "div", "label", "a", "br", "p", "b", "i", "del",
"strike", "u",
"img", "video", "audio", "iframe", "object", "embed", "param",
"blockquote",
"mark", "cite", "small", "ul", "ol", "li", "hr", "dl", "dt", "dd",
"sup", "sub",
"big", "pre", "code", "figure", "figcaption", "strong", "em",
"table", "tr", "td",
```

```
"th", "tbody", "thead", "tfoot", "h1", "h2", "h3", "h4", "h5",
"h6"]
```
You can set up another allowed tags on paste:

```
$('#redactor').redactor({
    allowedTags: ["a", "p", "b", "i"]
});
```

mobile
Default setting is `true`

This setting turns on/off the editor for mobile.

```
$('#redactor').redactor({
    mobile: false
});
```

protocol
Default setting is `http://`

This setting adds a protocol to the links.

```
$('#redactor').redactor({
    protocol: 'https://'
});
```
You can turn off adding a protocol.

```
$('#redactor').redactor({
    protocol: false
});
```

Creating Plugins

Plugins are here to extend Redactor's features and options. It's very easy to build a plugin.

Ok, here we go. Create a javascript file and name it, say, advanced.js. Now write the plugin:

```
if (typeof RedactorPlugins === 'undefined') var RedactorPlugins =
{};
```

```
RedactorPlugins.advanced = {

    your_method: function()
    {

    }
}
```

Now link your plugin on a page:

```
<!DOCTYPE html>
<html>
<head>
    <title>Redactor is awesome!</title>

    <meta charset="utf-8">

    <link rel="stylesheet" type="text/css" href="css/
yoursitesstyle.css" />
    <link rel="stylesheet" href="redactor/redactor.css" />

    <script type="text/javascript" src="lib/jquery-1.7.min.js"></
script>
    <script src="redactor/advanced.js"></script>
    <script src="redactor/redactor.js"></script>

    <script type="text/javascript">
    $(document).ready(
        function()
        {
            $('#redactor').redactor({
                plugins: ['advanced']
            });
        }
    );
    </script>

</head>
<body>
    <div id="page">
        <textarea id="redactor" name="content"></textarea>
    </div>
</body>
</html>
```

Plugin is turned on and all of Redactor's methods are available within it.

If you wish a particular method of your plugin to start with Redactor, you can do it like this with a plugin:

```
if (typeof RedactorPlugins === 'undefined') var RedactorPlugins =
{};

RedactorPlugins.advanced = {

    init: function()
    {
        alert(1);
    }
}
```

Obviously, each plugin may have several methods which may interact with each other:

```
if (typeof RedactorPlugins === 'undefined') var RedactorPlugins =
{};

RedactorPlugins.advanced = {

    init: function()
    {
        this.testAdvanced();
    },
    testAdvanced: function()
    {
        alert(2);
    }

}
```

You can start multiple plugins with Redactor:

```
<script type="text/javascript">
$(document).ready(
    function()
    {
        $('#redactor').redactor({
```

```
            plugins: ['fullscreen', 'advanced']
        });
    }
);
</script>
```

Plugins API

Selection and nodes

saveSelection
Save the cursor position.

```
somefunction: function()
{
    this.saveSelection();
}
```

restoreSelection
Restore the cursor position.

```
somefunction: function()
{
    this.restoreSelection();
}
```

getParentNode
Get parent element relative to the cursor.

```
somefunction: function()
{
    var node = this.getParentNode();
}
```

getCurrentNode
Get current element relative to the cursor.

```
somefunction: function()
{
    var node = this.getCurrentNode();
}
```

setFocusNode
Set the focus to a element.

```
somefunction: function()
{
    var node = $('<span class="new-element"></span>')[0];
    this.insertNodeAtCaret(node);
    this.setFocusNode(node);
}
```

getSelectedHtml
Get HTML selected fragment.

```
somefunction: function()
{
    var html = this.getSelectedHtml();
}
```

Insertion and code

insertNodeAtCaret
Insert node at the cursor.

```
somefunction: function()
{
    var node = $('<span class="new-element"></span>')[0];
    this.insertNodeAtCaret(node);
}
```

insertHtml
Insert HTML at the cursor.

```
somefunction: function()
{
    var html = $('<span class="new-element">code</span>');
    this.insertHtml(html);
}
```

getCode
Get the code of the editor.

```
somefunction: function()
{
```

```
    var html = this.getCode();
}
```

setCode
Paste the code into the editor.

```
somefunction: function()
{
    this.setCode('<p>html code</p>');
}
```

syncCode
Synchronize code editing layer and textarea.

```
somefunction: function()
{
    this.syncCode();
}
```

execCommand
This method allow you to execute the execCommand.

```
somefunction: function()
{
     this.execCommand('bold');
     this.execCommand('formatblock', '<h5>');
     this.execCommand('inserthtml', '<p>text</p>');
}
```

setBuffer
Before you insert something into the editor, you can turn the buffer. In this case, Undo will work.

```
somefunction: function()
{
    this.setBuffer();
}
```

Buttons

getBtn
Get object button on the key.

```
somefunction: function()
{
    var button = this.getBtn('bold');
}
```

setBtnActive
Make a button active.

```
somefunction: function()
{
   this.setBtnActive('bold');
}
```

setBtnInactive
Make a button inactive.

```
somefunction: function()
{
   this.setBtnInactive('bold');
}
```

inactiveAllButtons
Make all the buttons are inactive.

```
somefunction: function()
{
   this.inactiveAllButtons();
}
```

addBtnSeparator
Add a separator after all buttons.

```
somefunction: function()
{
   this.addBtnSeparator();
}
```

addBtnSeparatorAfter
Add a separator after a specific button.

```
somefunction: function()
{
   this.addBtnSeparatorAfter('bold');
}
```

addBtnSeparatorBefore
Add a separator before a specific button.

```
somefunction: function()
{
   this.addBtnSeparatorBefore('bold');
}
```

removeSeparatorAfter
Remove a separator after a specific button.

```
somefunction: function()
{
   this.removeSeparatorAfter('bold');
}
```

removeSeparatorBefore
Remove a separator before a specific button.

```
somefunction: function()
{
   this.removeSeparatorBefore('bold');
}
```

setBtnRight
Align button on the right edge.

```
somefunction: function()
{
   this.setBtnRight('bold');
}
```

setBtnLeft
Align button on the left edge.

```
somefunction: function()
{
   this.setBtnLeft('bold');
}
```

addBtn
Add a new button to the end of toolbar.

```
somefunction: function()
```

```
{    this.addBtn('newbutton', 'Title of button',
function(redactor_object, event, button_key) { // callback
function });
}
```

addBtnFirst
Add a new button to the start of toolbar.

```
somefunction: function()
{    this.addBtnFirst('newbutton', 'Title of button',
function(redactor_object, event, button_key) { // callback
function });
}
```

addBtnAfter
Add a new button after a specific button.

```
somefunction: function()
{    this.addBtnAfter('bold', 'newbutton', 'Title of button',
function(redactor_object, event, button_key) { // callback
function });
}
```

addBtnBefore
Add a new button before a specific button.

```
somefunction: function()
{    this.addBtnBefore('bold', 'newbutton', 'Title of button',
function(redactor_object, event, button_key) { // callback
function });
}
```

removeBtn
Remove a button.

```
somefunction: function()
{
   this.removeBtn('bold');
}
```

Modal

modalInit
Call a modal window.

```
somefunction: function()
{
    // callback (optional)
    var callback = $.proxy(function()
    {
        // some code
        this.saveSelection();

    }, this);

    // call a modal from element with ID #mymodal
    this.modalInit('Title of Modal', '#mymodal', 500, callback); //
500 it is the width of the window in pixels

    // call a modal with a code
    this.modalInit('Title of Modal', '<p>Your modal code</p>', 500,
callback);
}
```

modalClose
Close your modal window (all modal windows).

```
somefunction: function()
{
    this.modalClose();
}
```

Upload

uploadInit
Ajax upload.

```
// set field
<input type="file" name="file_name">

// call uploadInit with onchange event
somefunction: function()
{     this.uploadInit('file_name', { auto: true, url:
'your_upload_url', success: $.proxy(uploadCallback, this)  });
}
```

dragupload
Drag and drop upload.

```
// set field
<input type="file" id="file_id" name="file">

// call dragupload
somefunction: function()
{
    $('#file_id').dragupload(
    {
        url: 'your_upload_url',
        success: $.proxy(function(data)
        {
            fileUploadCallbackFunction(data);
        }, this)
    });
}
```

Utilities

normalize
Remove 'px' from string.

```
somefunction: function()
{
    var value = this.normalize('12px'); // return 12 as Number
}
```

isMobile
Detect mobile devices.

```
somefunction: function()
{
    var ismodal = this.isMobile(); // return true or false
}
```

oldIE
Detect IE less 9 version.

```
somefunction: function()
{
    var oldie = this.oldIE(); // return true or false
```

```
}
```

outerHTML
Get the element as HTML code.

```
somefunction: function()
{
    var html = this.outerHTML(element);
}
```