Ext.ux.form.TinyMCETextArea

ExtJS form field - a text area with integrated TinyMCE WYSIWYG editor.

Version: 2.6
Release date: 22.05.2013
ExtJS Version: 4.2.0
TinyMCE Version: 3.5.8
License: LGPL v2.1 or later, Sencha License
Author: Oleg Schildt

Implementation details

This component is a standard text area field, which is extended with the TinyMCE editor.

The TinyMCE editor is applied to the internal text area element of the Ext.form.field.TextArea.

Following tasks had to be solved:

- Time for starting of the editor initialization.
- Initialization in an initially invisible tab.
- Correct resizing of the editor by resizing of the underlying text in any layout.
- Enabling and disabling of the editor. Keeping of the cursor position by switching to the HTML text modus.
- Focusing of the editor field.
- Marking invalid.
- Activation and deactivation of the editor.
- Conflicts with the ExtJS standard css rules.

Initialization and Sizing

To correctly initialize the editor, we need the information of the actual size of the underlying text area. The size is not known at the time of the afterRender event. I add the 'resize' event handler only after rendering and start the initialization, when the first 'resize' event is fired. The initialization is done only once, in subsequent firing of the 'resize' event, only the adjustment of the editor size is done.

By sizing of the editor I faced the problem, that the editor occupies a little bit more height than it should. In usual case, it is unimportant, if the editor occupies a couple of pixels more, but in ExtJS, it should be exactly up to one pixel. So I had to do sizing by myself. The editor is a table with 2 or 3 rows.

- Tool bar
- Contents iframe
- Status bar (optional)

I retrieve the height of the tool bar and that of the status bar, then, I calculate the correct height for the contents iframe and set it.

To force the correct height already by the initialization and avoid the unbeautiful effect of flickering by size adjustment after initialization, I found out the following undocumented editor parameters:

- theme_advanced_row_height
- delta_height

Playing with them lets adjust the height, that is calculated by the TinyMCE while initialization, without adjusting the program code of the TinyMCE. I set them to

theme_advanced_row_height: 27
delta_height: 0

These values are good for the skin "extjs", which is made based on the blue skin "o2k7", and the doctype <!DOCTYPE html>.

The values may vary from skin to skin and doctype.

TinyMCETextArea in a tab control

If an element is a child of an invisible container like tab, there are a number of issues which make the life complicated:

- The element might not be completely rendered; it will be completely rendered only when the container becomes visible.
- The size of the element might be unknown until the container becomes visible. If you do size adjustment, while the container is not visible, the size might be calculated and set incorrectly.
- No show/hide event is fired for the element self if the container becomes visible or hidden. The event is fired only for that parent container, which is being actively shown or hidden.
- You have to attach the event handler to the correct parent container and do size adjustment only if the container becomes visible.

I check whether our TinyMCETextArea is a child of a tab panel. If yes, I attach the event handler to the tab change event and do size adjustment if the parent tab, which contains our TinyMCETextArea, becomes visible.

PS: It will not work if you have a tab control within another tab control!

Dynamic loading of the tiny_mce.js

If you do not include the tiny_mce.js code statically, but it is loaded dynamically only if it is required, you have to do some additional actions. You have to place somewhere in your code this initialization statement

```
window.tinyMCEPreInit = {
    suffix : '',
    base : '/js/tinymce' // your path to TinyMCE
};
```

This is necessary, because the TinyMCE is initialized upon "document ready" event, and in the case of dynamic loading, this event already occurred before and is not fired upon loading of the TinyMCE script. The author of the TinyMCE have provided this option – tinyMCEPreInit – for such cases.

Skin "extjs" and the native EXTJS windows for the editor inline popups

I created adjusted skins for the customizing of the inline popups. The customization concerns the skin for the main application and the skin for the online popups. The skins are named "extjs". They reside in the following directories:

tiny_mce/themes/advanced/skins/extjs/

tiny_mce/plugins/inlinepopups/skins/extjs/

The architecture of the TinyMCE popups lets override the standard functionality and use own windows for displaying TinyMCE popup dialogs like image insertion, link insertion, property editing etc.

I have introduced the class

Ext.ux.form.TinyMCETextAreaWindowManager

which overrides the standard tinymce.WindowManager and binds the native ExtJS windows to the TinyMCE popups dialogs.

Full screen mode

The full screen mode works as follows. A clone of the active editor is created and placed over all other elements with a very high z-index.

In order to enable the correct z-order of the popup windows in the full screen mode, the z-index of editor clone is decreased due to the current value of the ZIndexManger, so that subsequent popup windows are shown over the full screen editor clone.

Another important action is to assign the inyMCETextAreaWindowManager to the editor clone instead of the standard one.

Al these actions are done on the event "onExecCommand" if the command "mceFullScreen" has been fired.

Internationalization notes

Generally, no explicit internationalization is required for the component, the standard TinyMCE internationalization is enough. There are only two cases, where I use the texts which require internationalization. Since I have overridden the standard WindowManager to use the ExtJS windows for the TinyMCE popups dialogs, the alert box and confirm box need a title. I use the words "Information" and "Question".

```
alert: function(txt, cb, s) {
  Ext.MessageBox.alert(this.editor.getLang("Message", "Message"),
this.editor.getLang(txt,txt), function() {
    if (!Ext.isEmpty(cb)) {
      cb.call(this);
    }
  }, s);
},

confirm: function(txt, cb, s) {
  Ext.MessageBox.confirm(this.editor.getLang("Question", "Question"),
this.editor.getLang(txt,txt), function(btn) {
```

```
    if (!Ext.isEmpty(cb)) {
      cb.call(this, btn == "yes");
    }
  }, s);
},
```

If you want to be perfect by the internationalization, you have to provide the translations for the words "Information" and "Question" in the language files in the directory "lang". If you do not provide the translations, the English words will be used.

Setting and Getting Value

You can set and get value by using usual methods getValue() and setValue().

The editor does pre-formatting of the entered HTML text. It is important, for instance, to let the editor convert absolute or relative urls, make the entered HTML valid etc.

If you enter the HTML text manually in the HTML modus, the WYSIWYG editor does the pre-formatting also.

Important!

Since the text area is the master, and the TinyMCE Editor is an add-on, you have to call tinymce.triggerSave()

- before you get value with the method getValue()
- before you submit a form with that component

This call let all TinyMCE Editor Instances save the actual contents to the underlying text area.

Enabling and Disabling

The TinyMCE does not support disabling and enabling. I found out the following solution for disabling:

- I set the content iframe to content_editable=false.
- I remove all event listeners.
- I disable all tool bar buttons (there is a method for this).

By enabling:

- I set the content iframe to content_editable=true.
- I add all events again. I had luck that there is an internal method bindNativeEvents() that does all necessary things.
- I enable all tool bar buttons.
- I fire the event nodeChanged to adjust the enabled/disabled state of the buttons to the actual state of the editor. All buttons at once are never enabled.

Marking invalid

When the editor is active, the standard marking does not work. I had to add an invalid mark manually.

The best element of the editor for that purpose is the iframe container (a td element). Adding borders to that element does not harm sizing and positioning.

For marking invalid, I add the class "tinymce-error-field" to the iframe container.

Following rule should be written somewhere in a CSS file or style definition (best in the ui.css of the active editor skin):

```
.extjsSkin td.tinymce-error-field
{
  border: 1px dashed red;
}
```

It should be defined with a stronger rule than just

```
.tinymce-error-field {}
```

otherwise, the skin rules override this.

Changing CSS files for the contents on the fly

This is a very useful feature. The editor allows specifying of the CSS file, which will be applied for the contents. Let's assume a CMS. If the user changes the design template for the page that is opened for editing in the WYSWYG editor, the CSS file should be also changed and applied to the editable contents.

I implemented the method reinitEditor that does it. It is performed through removing the editor and initialization it again with the adjusted options.

Conflicts with ExtJS CSS

Following rule should be written somewhere in a CSS file or style definition (best in the ui.css of the active editor skin):

```
.x-border-box .mceEditor *
{
  box-sizing: content-box;
  -moz-box-sizing: content-box;
  -ms-box-sizing: content-box;
  -webkit-box-sizing: content-box;
}

.x-border-box .mceEditor .mceText
{
  background-color: white;
}
```

to override the ExtJS rule in ext-all.css:

```
.x-border-box *
{
  box-sizing: border-box;
  -moz-box-sizing: border-box;
  -ms-box-sizing: border-box;
  -webkit-box-sizing: border-box;
}
```

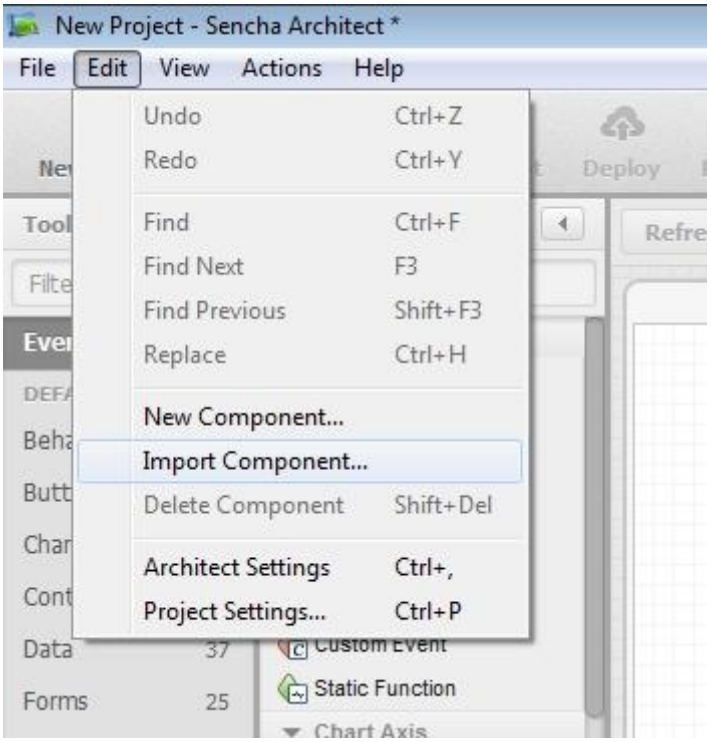
Otherwise the dropdown lists of the editor tool bars are rendered wrong, to smaller height:

Whereas, the correct way is:

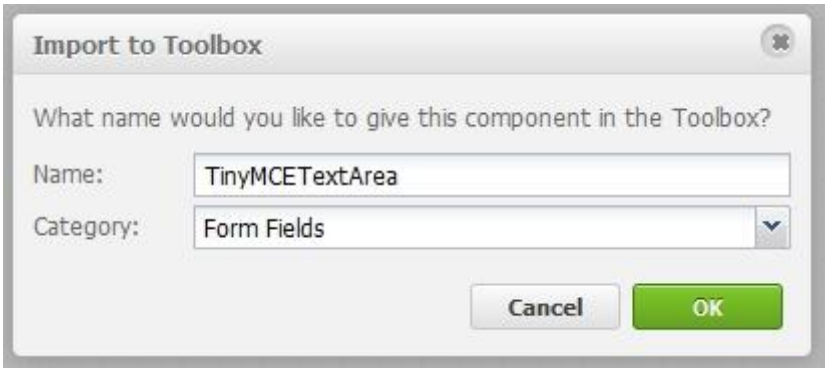
Integration into the Sencha Architect

The file TinyMCETextArea.xdc, kindly offered by Gregory, is included in the package of the TinyMCETextArea.

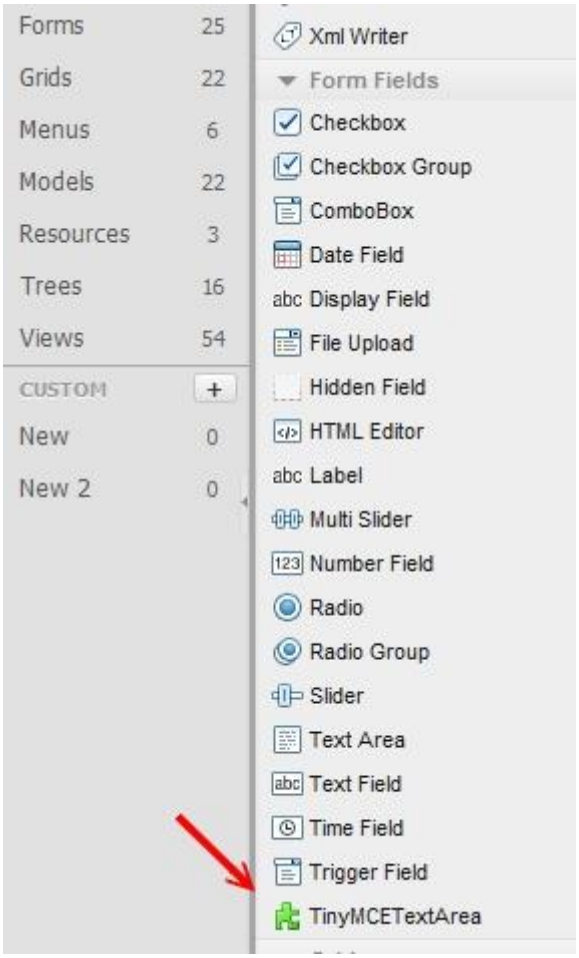
Open the Sencha Architect and in the menu "Edit" select "Import Component". Select the file "TinyMCETextArea.xdc":



Choose the category of the component and provide the name for the component:



The component should have been added:



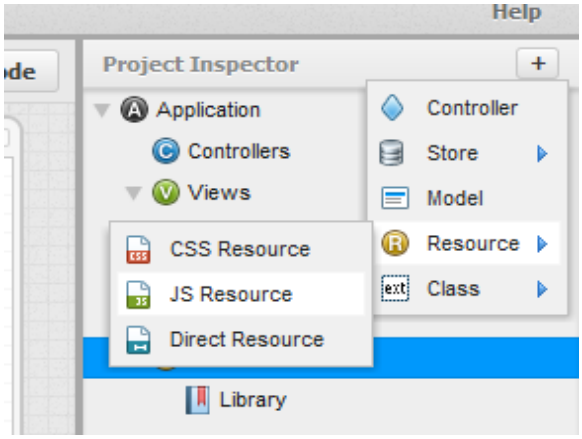
Attention!

For some reason, the component is visible in the "Form Fields" branch only over the general view. If you click "Forms", the component is not listed there.

Now, we create the project and copy the following files and directories to the project directory:

- tinymce
- TinyMCETextArea.js

Next, add the following JS files to the project resources:

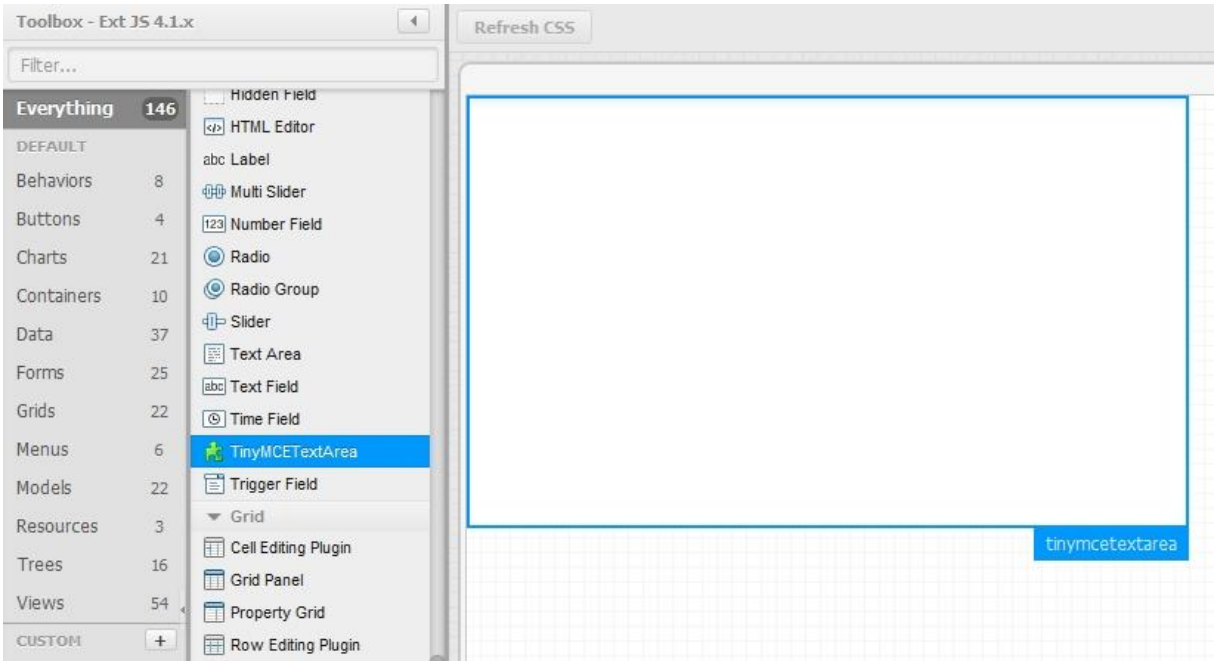


1. tinymce\tiny_mce_src.js
2. TinyMCETextArea.js

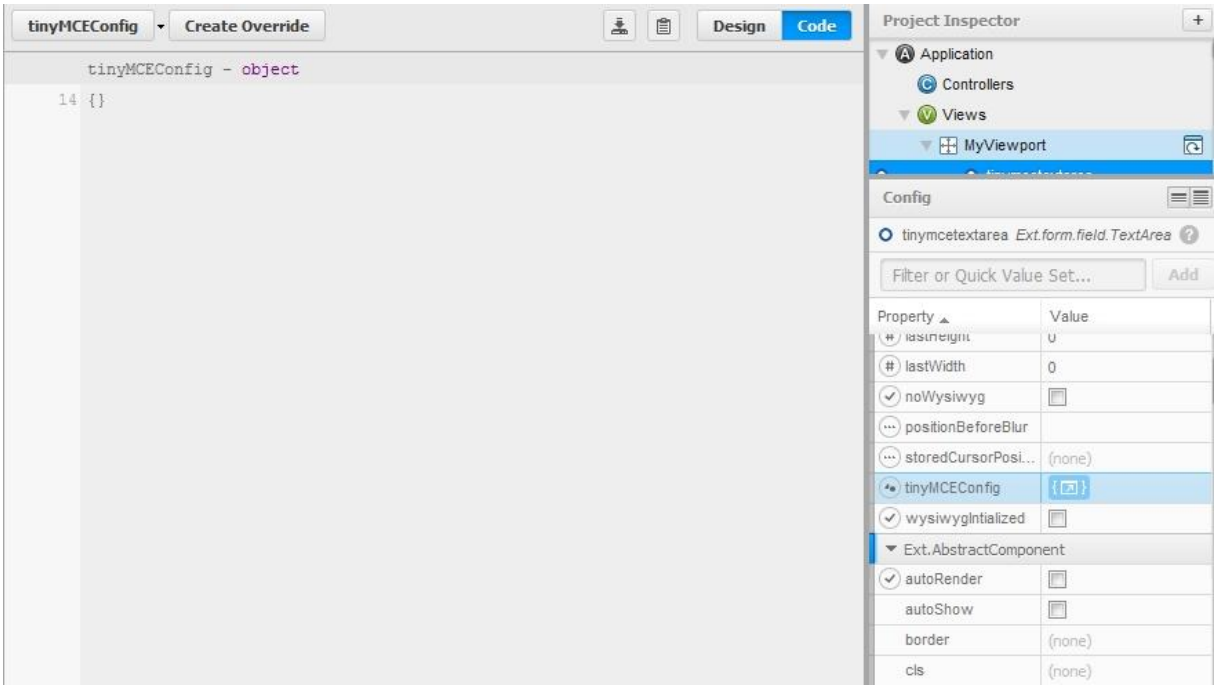
Attention!

The order of the file including is important!

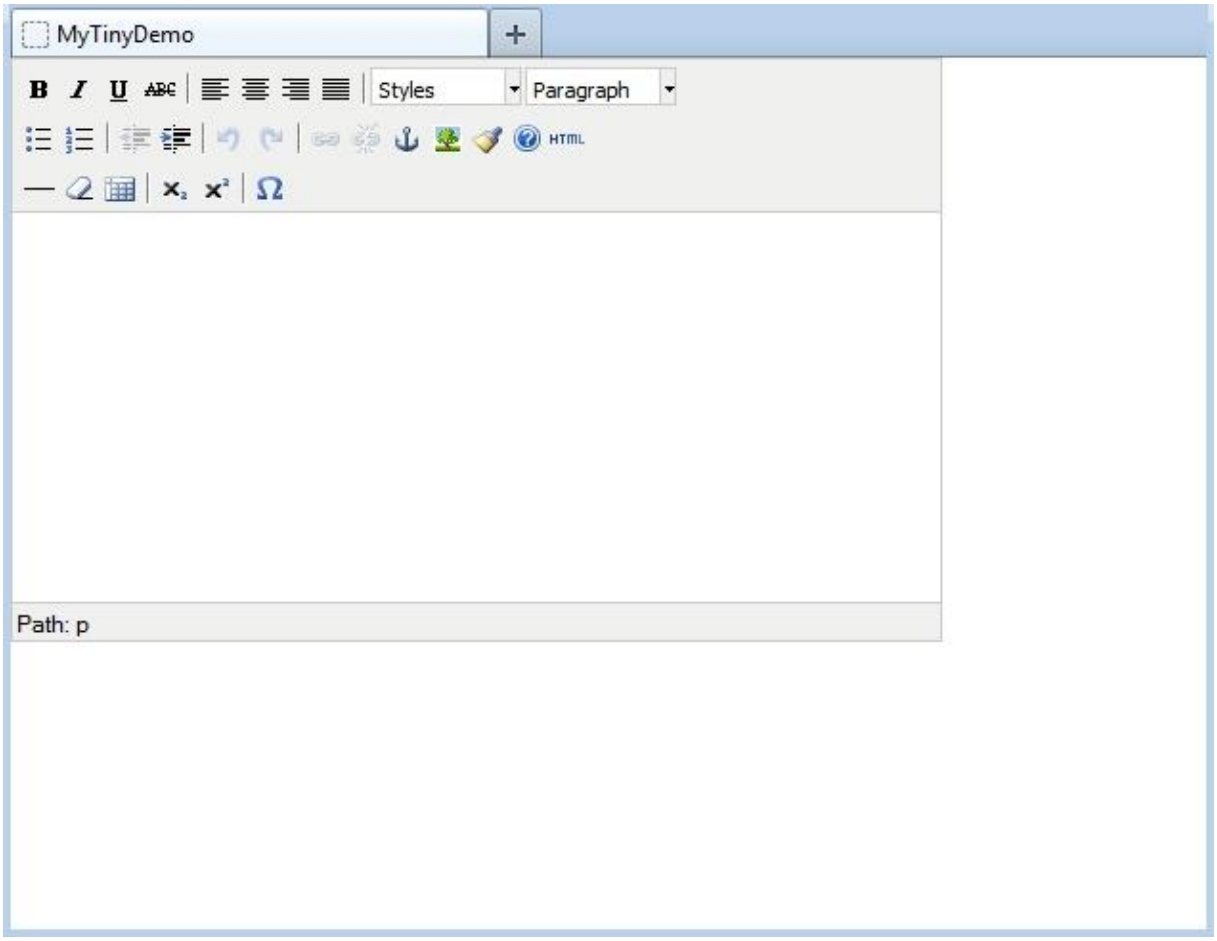
Put the component to the desired place:



Set the tinyMCEConfig property for the component.



Now, we can compile the project and see the result:



Reference

Generally, the component derives all properties and methods from the TextArea. Here, only the new and overridden properties and methods are described.

Properties

noWysiwyg

This property forces that the editor is not initially active.

disabled

This property let make the control initially disabled.

readOnly

This property let make the control initially readOnly.

tinyMCEConfig

This property contains the TinyMCE settings which are passed to the editor by the initialization.

Methods

enable([Boolean silent])

Enable the component

Parameters:

silent : Boolean (optional)

Passing true will supress the 'enable' event from being fired.

Defaults to: false

disable([Boolean silent])

Disable the component

Parameters:

silent : Boolean (optional)

Passing true will supress the 'disable' event from being fired.

Defaults to: false

setReadOnly([Boolean readOnly])

Set the control ReadOnly or not ReadOnly.

Parameters:

readOnly : Boolean

The desired state of the control.

showEditor()

Activate the editor, WYSIWYG modus.

hideEditor()

Deactivate the editor, HTML text modus.

toggleEditor()

Toggle the modus WYSIWYG versus HTML text.

isEditorHidden()

Return whether the WYSIWYG editor is hidden or not.

removeEditor()

Completely remove the WYSIWYG editor. The control becomes a normal text area.

focus([Boolean selectText], [Boolean/Number delay]) : Ext.Component

Try to focus this component.

Parameters:

selectText : Boolean (optional)

If applicable, true to also select the text in this component

delay : Boolean/Number (optional)

Delay the focus this number of milliseconds (true for 10 milliseconds).

Returns:

Ext.Component (this)

storeCurrentSelection()

Store the current cursor position or the current selection for later restoring. This method should be called before calling any dialog for inserting of a placeholder or other text into the current cursor position or instead of the current selection.

restoreCurrentSelection()

Restore the cursor position or the current selection for later text insertion. This method should be called immediately before inserting of a placeholder or other text into the current cursor position or instead of the current selection.

insertText(txt)

Insert the text "txt" at the current cursor position or instead of the current selection.

getValue() : Object

Returns:

The actual HTML text

setValue(Object value) : Ext.ux.form.TinyMCETextArea

Sets a data value into the field and runs the change detection and validation. The value is pre-formatted with the WYSIWYG editor.

Parameters:

value : Object

The value to set

Returns:

Ext.ux.form.TinyMCETextArea (this)

reinitEditor([cfg]) : Ext.ux.form.TinyMCETextArea

Reinitialize the editor with the adjusted settings.

Parameters:

cfg : Object (optional)

The configuration options which differ from the initial option. All options do not need be repeated.

Returns:

Ext.ux.form.TinyMCETextArea (this)