WYMeditor Documentation

Release 1.0.0b5

Jean-Francois Hovinne

November 05, 2013

Contents

Contents:

CHAPTER 1

General

1.1 About WYMeditor

WYMeditor is an open source web-based WYSIWYM editor with semantics and standards in mind. The WYM-part stands for "What You Mean" compared to the more common "What You See Is What You Get".

WYMeditor's main concept is to leave details of the document's visual layout, and to concentrate on its structure and meaning, while trying to give the user as much comfort as possible (at least as WYSIWYG editors).

WYMeditor has been created to generate perfectly structured XHTML strict code, to conform to the W3C specifications and to facilitate further processing by modern applications.

With WYMeditor, the code can't be contaminated by visual informations like font styles and weights, borders, colors, ... The end-user defines content meaning, which will determine its aspect by the use of style sheets. The result is easy and quick maintenance of information.

As the code is compliant to W3C specifications, you can for example process it using a XSLT (at the client or the server side), giving you a wide range of applications.

For further explanations, please see http://www.wymeditor.org/.

1.1.1 Why WYMeditor?

WYMeditor is different from the traditional editors because we are 100% focused on providing a simple experience for users that separates the content of their document from the presentation of that document. We focus on enforcing web standards and separating a document's structure (HTML) from its presentation (CSS). Your users won't know and shouldn't care about HTML, but when they need consistent, standards-compliant, clean content, they'll thank you.

There are lots of choices when it comes to a brower-based editor and many of them are stable, mature projects with thousands of users. If you need an editor that gives total control and flexibility to the user (not you, the developer), then WYMeditor is probably not for you. If you want an editor that you can customize to provide the specific capabilities your users need, and you want users focused on the structure of their content instead of tweaking fonts and margins, you should give WYMeditor a try.

We also fully support Internet Explorer 6, for those poor souls who have no choice.

1.1.2 Compatibility

WYMeditor is compatible with:

- IE 6, 7, 8 and 9
- Firefox 3.6.x, 9.x and 10.x
- Opera 9.5+
- Safari 3.0+
- Google Chrome Stable and Beta

1.1.3 Copyright

1.2 Getting Help

- **Wiki/Docs:** https://github.com/wymeditor/wymeditor/wiki
- **Forum:** http://community.wymeditor.org
- **Issue tracking:** https://github.com/wymeditor/wymeditor/issues
- **Official branch:** https://github.com/wymeditor/wymeditor

To Read more on contributing, see *Contributing*.

1.3 Coding Standard

The goal of this document is to define a set of general rules regarding the formatting and structure of the WYMeditor code as well as defining some best practices. It should also serve as a god starting point for developers that want to contribute code to the WYMeditor project.

1.3.1 jslint

All Javascript source should pass the latest stable version of JSlint using the options:

```
jslint --es5 false --white --indent 4
```

1.3.2 Crockford Javascript Code Conventions

Please refer to the Crockford Javascript Code Conventions for a set of general rules. The points listed below are not included in Crockford Conventions and/or specific to the WYMeditor project.

1.3.3 Formatting and Style

Naming Conventions

Variables and Functions

Give variables and function **meaningful names**. Use mixedCase (lower CamelCase) for names spanning several words. "Constants" should be in all CAPITAL_LETTERS with underscores to separate words. Avoid the use of Hungarian Notation, instead make sure to "type" your variables by assigning default values and/or using comments.

Example:

```
var elements = [];
var VERSION = 0.6;
function parseHtml () {};
```

Constructors

Constructors should be named using PascalCase (upper CamelCase) for easier differentiation.

Example:

```
function MyObject () {}
MyObject.prototype = {
function myMethod () {}
}
```

Event Handlers

Prepend "on" to the event handler name for easier differentiation.

Example:

```
function onEventName (event) {}
```

Namespacing

All code should be placed under the WYMeditor namespace to avoid creating any unnecessary global variables. If you're extending and/or modifying WYM, place you code where you see fit (most likely WYMeditor.plugins).

WYMeditor.core contains the Editor object and the SAPI as well as HTML, CSS and DOM parsers which make out the core parts of WYMeditor.

WYMeditor.ui contains the UI parts of WYM (i.e. the default Toolbar and Dialogue objects).

WYMeditor.util contains any utility methods or objects, see *Leave the Natives Alone*.

WYMeditor.plugins – place your plug-ins here.

1.3.4 Inheritance and "Classes"

There's a lot of different ways of doing inheritance in JavaScript. There have been attempts to emulate Classes and several patterns trying enhance, hide or modify the prototypal nature of JavaScript – some more successful than others.

But in order to keep things familiar for as many JavaScript developers as possible we're sticking with the "Pseudo Classical" model (constructors and prototypes).

It's not that the different variations of the "Pseudo Classical" model out there are all bad, but there is no other "standard" way of doing inheritance.

Private members

Add description

Events

Add description

1.3.5 Other Rules and Best Practices

Leave the Natives Alone

WYMeditor is used by a lot of people in a lot of different environments thus modifying the prototypes for native objects (such as Array or String) can result in unwanted and complicated conflicts.

The solution is simple – simply leave them alone. Place any kind of general helper methods under WYMeditor.util.

Use Literals

This is a basic one – but there's still a lot of developers that use the Array and Object constructors.

http://yuiblog.com/blog/2006/11/13/javascript-we-hardly-new-ya/

Use the `which` Property of jQuery Event Objects

When watching for keyboard key input, use the `event.which` property to find the inputted key instead of `event.keyCode` or `event.charCode`. This should be done for consistency across the project because the `event.which` property normalizes `event.keyCode` and `event.charCode` in jQuery. Using `event.which` is also the recommended method by jQuery for watching keyboard key input.

Further Reading

Got any other links that you think can be of help for new WYM developers? Share them here!

- http://dev.opera.com/articles/view/javascript-best-practices/

1.4 WYMeditor in the Wild

There was a list on the old site which should be migrated here, making it easier to keep it relevant and up to date. Please migrate or add links to working integrations.

1.4.1 Open source

3rd-party Skins

- wymeditor-refine A skin for WYMEditor extracted and tweaked from Refinery CMS

Content Management Systems

- django CMS
- Drupal, via the Wysiwyg module
- Frog CMS, plugin available here
- MediaWiki
- Midgard CMS Framework
- Refinery CMS, also available as a hosted version

Frameworks

- Jelix

Other

- Adminer single-file database manager in PHP

1.4.2 Commercial

- PolicyStat, policy and procedure management for healthcare
- Refinery a hosted CMS

CHAPTER 2

Version 1.0 (and 0.5)

2.1 Getting Started

2.1.1 Running the Examples Locally

Each release of WYMeditor comes bundled with a set of examples (available online here). To run these locally, they need to be served on a proper web server. If not, WYMeditor might not work properly as most modern browsers block certain JavaScript functionality for local files.

To test WYMeditor without setting up something like Apache, Nginx or uploading the code to a remote server, open your wymeditor directory in a terminal and run:

```
$ python -m SimpleHTTPServer
```

Note: On Windows: if the Python executable is not already on your PATH, replace "python" with the absolute path to your python.exe, usually something like "C:Python2Xpython.exe".

This will start a simple server in your current working directory on port 8000. If you already have something else running on that port, an optional port number can be provided as the last argument.

Now point your browser to `http://localhost:8000/` and browse to the examples directory.

Done!

2.1.2 Setting up WYMeditor

1. WYMeditor requires a version of jQuery between 1.3.x and 1.9.x. First ensure that your page includes jQuery.

   Note: If a version of jQuery at or above 1.8.0 is used, WYMeditor also requires jQuery Migrate. Ensure that your page also includes jQuery Migrate after jQuery is included. See *Page Integration with jQuery Migrate*.

2. Download the Version 1.0.0b4 archive and extract the contents to a folder in your project.
3. Include the `wymeditor/jquery.wymeditor.min.js` file on your page. This file will pull in anything else that's required.

```
<script type="text/javascript" src="/wymeditor/jquery.wymeditor.min.js"></script>
```

4. Now use the `wymeditor()` function to select one of your `textarea` elements and turn it in to a WYMeditor instance. eg. if you have a `textarea` with the class `my-wymeditor`:

```
$('.my-wymeditor').wymeditor();
```

Note: You'll probably want to do this initialization inside a `$(document).ready()` block.

---

5. If you'd like to receive the valid HTML your editor produces on form submission, just add the class wymupdate to your submit button.

```
<input type="submit" class="wymupdate" />
```

6. ???
7. Profit!

Sample Minimal Page Integration

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>WYMeditor</title>
<script type="text/javascript" src="jquery/jquery.js"></script>
<script type="text/javascript" src="wymeditor/jquery.wymeditor.pack.js"></script>
<script type="text/javascript">

jQuery(function() {
    jQuery(".wymeditor").wymeditor();
});

</script>
</head>

<body>
<form method="post" action="">
<textarea class="wymeditor"></textarea>
<input type="submit" class="wymupdate" />
</form>
</body>

</html>
```

Explanation

- `jQuery(function() {});` is a shorthand for `jQuery(document).ready()`, allowing you to bind a function to be executed when the DOM document has finished loading. If you prefer, you can use the jQuery '$' shortcut.
- `jQuery(".wymeditor").wymeditor();` will replace every element with the class `wymeditor` by a WYMeditor instance.
- The value of the element replaced by WYMeditor will be updated by e.g. clicking on the element with the class `wymupdate`. See *Customization Options*.

More examples with different plugins and configuration options can be found in your `examples` directory.

Page Integration with jQuery Migrate

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>WYMeditor</title>
<script type="text/javascript" src="jquery/jquery.js"></script>

<!-- Include jQuery Migrate after jQuery -->
<script type="text/javascript" src="jquery/jquery-migrate.min.js"></script>

<script type="text/javascript" src="wymeditor/jquery.wymeditor.pack.js"></script>
<script type="text/javascript">

jQuery(function() {
    jQuery(".wymeditor").wymeditor();
});

</script>
</head>

<body>
<form method="post" action="">
<textarea class="wymeditor"></textarea>
<input type="submit" class="wymupdate" />
</form>
</body>

</html>
```

This is only necessary if the included version of jQuery is at or above 1.8.0. Be sure to include jQuery Migrate **after** including jQuery.

2.1.3 Customization Options

Since 0.4, getting started examples are integrated in WYMeditor's package. See the 'examples' directory.

```
$(jqexpr).wymeditor(options)
```

This function will be applied to elements matching a jQuery expression `jqexpr`.

Example: Replaces each `textarea` having the class `wymeditor` by a WYMeditor instance:

```
$("textarea.wymeditor").wymeditor();
```

Parameters

The function accepts an optional parameter: an array of `options`.

Example: Initializes WYMeditor HTML content and displays an alert box when ready:

```
$(".wymeditor").wymeditor({

  //options
  html: "<p>test<\/p>",
  postInit: function() {
```

```
    alert('OK');
  }
});
```

List of Options

Check the file jquery.wymeditor.js, function `$j.fn.wymeditor`, for the complete list of options and their default values.

html

Initializes the editor's HTML content.

Example:

```
html: "<p>Hello, World!<\/p>"
```

Note: In Javascript, it's a good habit to escape forward slashes, like this: `<\/p>`.

---

basePath

WYMeditor's relative/absolute base path (including the trailing slash), used while loading the iframe and the dialogs.

Example:

```
basePath: "/admin/edit/wymeditor/"
```

This value is automatically guessed by `computeBasePath`, which looks for the `script` element loading jquery.wymeditor.js.

skinPath

WYMeditor skin (theme) path, used to load the skin.

Example:

```
skinPath: "wymeditor/skins/default/"
```

This value is automatically guessed, based on the `basePath` value.

wymPath

WYMeditor main JS file path.

This value is automatically guessed by `computeWymPath`, which looks for the `script` element loading jquery.wymeditor.js.

iframeBasePath

WYMeditor iframe base path.

This value is automatically guessed, based on the `basePath` value.

jQueryPath

jQuery JS file path.

Example:

```
jQueryPath: "/js/jquery.js"
```

This value is automatically guessed by `computeJqueryPath`, which looks for the `script` element loading jquery.js.

lang

The language to use with WYMeditor. Default is English (en). Codes are in ISO-639-1 format.

Language packs are stored in the `wymeditor/lang` directory.

How to use a Custom Language

Just initialize the option:

```
$('.wymeditor').wymeditor({ lang: 'pl' });
```

boxHtml

The editor container's HTML. This option allows you to customize the HTML containing a WYMeditor instance.

logoHtml

The WYMeditor logo HTML. This option allows you to customize the HTML which displays the WYMeditor logo.

If you prefer to hide the WYMeditor logo, use an empty string:

```
$('.wymeditor').wymeditor({ logoHtml: '' });
```

In such a case, please consider making a donation to the project.

iframeHtml

The iframe (used for editing) container's HTML.

styles & stylesheet

Allows you to easily configure the editor's styles. Advantageously replaces `editorStyles`, `dialogStyles` and `classesItems`.

Define the styles using the `styles` option OR point to an external stylesheet, using the `stylesheet` option.

Example, using `styles`:

```
styles:
      '/* PARA: Date */                                          '+
      '.date p{                                                  '+
      '  color: #ccf;                                            '+
      '  /* background-color: #ff9; border: 2px solid #ee9; */   '+
      '}                                                         '+
      '                                                          '+
      '/* PARA: Hidden note */                                   '+
      '.hidden-note p /* p[@class!="important"] */ {             '+
      '     display: none;                                       '+
      '    /* color: #999; border: 2px solid #ccc; */           '+
      '}                                                         '
```

Example, using `stylesheet`:

```
$('.wymeditor').wymeditor({ stylesheet: 'stylesheet.css' });
```

Use this example stylesheet as a reference.

editorStyles

An array of classes, applied on the editor's content, in the form of: `{'name': 'value', 'css': 'value'}`

Example:

```
editorStyles: [
  {'name': '.hidden-note', 'css': 'color: #999; border: 2px solid #ccc;'},
  {'name': '.border', 'css': 'border: 4px solid #ccc;'}
]
```

toolsHtml

The tools panel's HTML.

toolsItemHtml

The tools buttons' HTML template.

toolsItems

An array of tools buttons, inserted in the tools panel, in the form of: `{'name': 'value', 'title': 'value', 'css': 'value'}`

Example:

```
toolsItems: [
  {'name': 'Bold', 'title': 'Strong', 'css': 'wym_tools_strong'},
  {'name': 'Italic', 'title': 'Emphasis', 'css': 'wym_tools_emphasis'}
]
```

Default value:

```
toolsItems: [
  {'name': 'Bold', 'title': 'Strong', 'css': 'wym_tools_strong'},
  {'name': 'Italic', 'title': 'Emphasis', 'css': 'wym_tools_emphasis'},
  {'name': 'Superscript', 'title': 'Superscript', 'css': 'wym_tools_superscript'},
  {'name': 'Subscript', 'title': 'Subscript', 'css': 'wym_tools_subscript'},
  {'name': 'InsertOrderedList', 'title': 'Ordered_List', 'css': 'wym_tools_ordered_list'},
  {'name': 'InsertUnorderedList', 'title': 'Unordered_List', 'css': 'wym_tools_unordered_list'},
  {'name': 'Indent', 'title': 'Indent', 'css': 'wym_tools_indent'},
  {'name': 'Outdent', 'title': 'Outdent', 'css': 'wym_tools_outdent'},
  {'name': 'Undo', 'title': 'Undo', 'css': 'wym_tools_undo'},
  {'name': 'Redo', 'title': 'Redo', 'css': 'wym_tools_redo'},
  {'name': 'CreateLink', 'title': 'Link', 'css': 'wym_tools_link'},
  {'name': 'Unlink', 'title': 'Unlink', 'css': 'wym_tools_unlink'},
  {'name': 'InsertImage', 'title': 'Image', 'css': 'wym_tools_image'},
  {'name': 'InsertTable', 'title': 'Table', 'css': 'wym_tools_table'},
  {'name': 'Paste', 'title': 'Paste_From_Word', 'css': 'wym_tools_paste'},
  {'name': 'ToggleHtml', 'title': 'HTML', 'css': 'wym_tools_html'},
  {'name': 'Preview', 'title': 'Preview', 'css': 'wym_tools_preview'}
]
```

containersHtml

The containers panel's HTML.

containersItemHtml

The containers buttons' HTML template.

containersItems

An array of containers buttons, inserted in the containers panel, in the form of: `{'name': 'value', 'title': 'value', 'css': 'value'}`

Example:

```
containersItems: [
  {'name': 'P', 'title': 'Paragraph', 'css': 'wym_containers_p'},
  {'name': 'H1', 'title': 'Heading_1', 'css': 'wym_containers_h1'}
]
```

classesHtml

The classes panel's HTML.

classesItemHtml

The classes buttons' HTML template.

classesItems

An array of classes buttons, inserted in the classes panel, in the form of: `{'name': 'value', 'title': 'value', 'expr': 'value'}`, where `expr` is a jQuery expression.

Example:

```
classesItems: [
  {'name': 'date', 'title': 'PARA: Date', 'expr': 'p'},
  {'name': 'hidden-note', 'title': 'PARA: Hidden note', 'expr': 'p[@class!="important"]'}
]
```

In this example, the class `date` can be applied on paragraphs, while the class `hidden-note` can be applied on paragraphs without the class `important`.

statusHtml

The status bar's HTML.

htmlHtml

The HTML box's HTML.

Selectors

WYMeditor uses jQuery to select elements of the interface. You'll need these options if you e.g. customize the panels' HTML.

- boxSelector
- toolsSelector
- toolsListSelector
- containersSelector
- classesSelector
- htmlSelector
- iframeSelector
- statusSelector
- toolSelector
- containerSelector
- classSelector
- htmlValSelector
- hrefSelector

- srcSelector
- titleSelector
- altSelector
- textSelector
- rowsSelector
- colsSelector
- captionSelector
- submitSelector
- cancelSelector
- previewSelector
- dialogLinkSelector
- dialogImageSelector
- dialogTableSelector
- dialogPasteSelector
- dialogPreviewSelector
- updateSelector

Example:

```
classesSelector: ".wym_classes"
```

updateSelector & updateEvent

Allows you to update the value of the element replaced by WYMeditor (typically a `textarea`) with the editor's content while e.g. clicking on a button in your page.

updateSelector is a jQuery expression, updateEvent is a jQuery event.

Example:

```
updateSelector: ".my-submit-button",
updateEvent:    "click"
```

dialogFeatures

The dialogs' features.

Example:

```
dialogFeatures: "menubar=no,titlebar=no,toolbar=no,resizable=no,width=560,height=300,top=0,left=0"
```

dialogHtml

The dialogs' HTML template.

dialogLinkHtml

The link dialog's HTML template.

dialogImageHtml

The image dialog's HTML template.

dialogTableHtml

The table dialog's HTML template.

dialogPasteHtml

The 'Paste from Word' dialog's HTML template.

dialogPreviewHtml

The preview dialog's HTML template.

dialogStyles

An array of classes, applied to the dialogs, in the form of: `{'name': 'value', 'css': 'value'}`

skin

The skin you want to use.

Example:

```
skin: 'custom'
```

stringDelimiterLeft & stringDelimiterRight

WYMeditor uses a simple function to replace strings delimited by these two strings for e.g. the l10n system.

preInit(wym)

A custom function which will be executed once, before WYMeditor's initialization.

Parameters:

- wym: the WYMeditor instance

preBind(wym)

A custom function which will be executed once, before binding handlers on events (e.g. buttons click).

Parameters:

- wym: the WYMeditor instance

postInit(wym)

A custom function which will be executed once, when WYMeditor is ready.

Parameters:

- wym: the WYMeditor instance

Example:

```
postInit: function(wym) {
  //activate the 'tidy' plugin, which cleans up the HTML
  //'wym' is the WYMeditor instance
  var wymtidy = wym.tidy();
  wymtidy.init();
}
```

preInitDialog(wym,wdw)

A custom function which will be executed before a dialog's initialization.

Parameters:

- wym: the WYMeditor instance
- wdw: the dialog's window object

postInitDialog(wym,wdw)

A custom function which will be executed when a dialog is ready.

Parameters:

- wym: the WYMeditor instance
- wdw: the dialog's window object

Basic Customization Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>WYMeditor</title>
<script type="text/javascript" src="jquery/jquery.js"></script>
<script type="text/javascript" src="wymeditor/jquery.wymeditor.pack.js"></script>
<script type="text/javascript">

jQuery(function() {
    jQuery(".wymeditor").wymeditor({
       html: '<p>Hello, World!<\/p>',
       stylesheet: 'styles.css'
    });
});

</script>
</head>

<body>
<form method="post" action="">
<textarea class="wymeditor"></textarea>
<input type="submit" class="wymupdate" />
</form>
</body>

</html>
```

Explanation

- The `html` option will initialize the editor's content.
- The `stylesheet` option will automagically parse your CSS file to populate the Classes panel and to initialize the visual feedback.

2.1.4 API

Note: In the code examples below, `wym` is a variable which refers to the WYMeditor instance, and which must be initialized.

---

box

Return the WYMeditor container.

html(sHtml)

Get or set the editor's HTML value.

Example:

```
wym.html("<p>Hello, World.</p>");
```

xhtml

Get the cleaned up editor's HTML value.

exec(cmd)

Execute a command.

Supported command identifiers

- Bold: set/unset `strong` on the selection
- Italic: set/unset `em` on the selection
- Superscript: set/unset `sup` on the selection
- Subscript: set/unset `sub` on the selection
- InsertOrderedList: create/remove an ordered list, based on the selection
- InsertUnorderedList: create/remove an unordered list, based on the selection
- Indent: 'indent' the list element
- Outdent: 'outdent' the list element
- Undo: undo an action
- Redo: redo an action
- CreateLink: open the link dialog and create/update a link on the selection
- Unlink: remove a link, based on the selection
- InsertImage: open the image dialog and insert/update an image
- InsertTable: open the table dialog and insert a table
- Paste: opens the paste dialog and paste raw paragraphs from an external application, e.g. Word
- ToggleHtml: show/hide the HTML value
- Preview: open the preview dialog

paste(data)

Parameters

- data: string

Description

Paste raw text, inserting new paragraphs.

insert(data)

Parameters

- data: XHTML string

Description

Insert XHTML string at the cursor position. If there's a selection, it is replaced by `data`.

Example:

```
wym.insert('<strong>Hello, World.</strong>');
```

wrap(left, right)

Parameters

- left: XHTML string
- right: XHTML string

Description

Wrap the inline selection with XHTML.

Example:

```
wym.wrap('<span class="city">', '</span>');
```

unwrap()

Unwrap the selection, by removing inline elements but keeping the selected text.

container(sType)

Get or set the selected container.

Example: switch the container to Heading 1.

```
wym.container('H1');
```

Example: get the selected container.

```
wym.status(wym.container().tagName);
```

toggleClass(sClass, jqexpr)

Set or remove the class `sClass` on the selected container/parent matching the jQuery expression `jqexpr`.

Example: set the class `my-class` on the selected paragraph with the class `my-other-class`.

```
wym.toggleClass('.my-class', 'P.my-other-class')
```

status(sMessage)

Update the HTML value of WYMeditor' status bar.

Example:

```
wym.status("This is the status bar.");
```

update

Update the value of the element replaced by WYMeditor and the value of the HTML source textarea.

dialog(sType)

Open a dialog of type `sType`.

Supported values: Link, Image, Table, Paste_From_Word.

Example:

```
wym.dialog('Link');
```

toggleHtml

Show/hide the HTML source.

replaceStrings(sVal)

Localize the strings included in `sVal`.

encloseString(sVal)

Enclose a string in string delimiters.

Custom jQuery properties

jQuery.wymeditors(i)

Returns the WYMeditor instance with index i (zero-based).

Example:

```
jQuery.wymeditors(0).toggleHtml();
```

2.1.5 Plugins

The WYMeditor includes a simple plugin which demonstrates how easy it is to write plugins for WYMeditor. See *Plugin System Architecture* for more details.

Note: Next versions will use a more advanced events handling architecture which will allow a better interaction between plugins and the editor.

Using a Plugin

To use a plugin you need to include it's script file using a `<script>` tag and then initialize it (passing an instance of wym) in the `postInit` function, passed as an option when you call `$().wymeditor()`.

```
postInit: function(wym) {
    //activate the 'tidy' plugin, which cleans up the HTML
    //'wym' is the WYMeditor instance
    var wymtidy = wym.tidy();
    // You may also need to run some init functions on the plugin, however this depends on the plugi
    wymtidy.init(wym);
}
```

Writing Plugins

Writing a plugin for WYMeditor is quite easy, if you have a basic knowledge of jQuery and Object Oriented Programming in Javascript.

Once you decide the name for your plugin you should create a folder of that name in the plugins folder and then a file called jquery.wymeditor.__plugin_name__.js. You need to include this file in your HTML using a `<script>` tag.

For details on interacting with the editor, including the selection, see *API*.

Example Plugin

```
// wymeditor/plugins/hovertools/jquery.wymeditor.hovertools.js
// Extend WYM
Wymeditor.prototype.hovertools = function() {
    var wym = this;

    //bind events on buttons
    $j(this._box).find(this._options.toolSelector).hover(
        function() {
        wym.status($j(this).html());
        },
        function() {
        wym.status(' ');
        }
    );
};
```

This example extends WYMeditor with a new method `hovertools`, and uses jQuery to execute a function while the mouse hovers over WYMeditor tools.

`this._box` is the WYMeditor container, `this._options.toolSelector` is the jQuery selector, `wym.status()` displays a message in the status bar.

Adding a Button to the Tool Bar

```
// Find the editor box
var wym = this,
    $box = jQuery(this._box);

//construct the button's html
var html = '' +
    "<li class='wym_tools_fullscreen'>" +
        "<a name='Fullscreen' href='#' " +
            "style='background-image: url(" +
                wym._options.basePath +
                "plugins/fullscreen/icon_fullscreen.gif)'>" +
            "Fullscreen" +
        "</a>" +
    "</li>";
//add the button to the tools box
$box.find(wym._options.toolsSelector + wym._options.toolsListSelector)
    .append(html);
```

(work in progress)

Available Plugins

See *Plugins* for a listing and descriptions of the plugins included in the download and available third party plugins.

2.2 Developers

2.2.1 Building WYMeditor

1. Get a copy of the source using git:

```
$ git clone git://github.com/wymeditor/wymeditor.git
```

2. Install `make`, Node.js and UglifyJS. To install UglifyJS using NPM run the following:

```
$ npm install -g uglify-js
```

3. Run `make` from your git clone:

```
$ cd wymeditor
$ make
```

The resulting compressed distribution will appear in your `dist` directory.

Building with Google's Closure Compiler (Java)

The default WYMeditor distribution is built with UglifyJS, which requires the installation of Node.js. If you prefer Java and/or Google's Closure Compiler, you can follow these instructions instead.

1. Get a copy of the source using git:

```
$ git clone git://github.com/wymeditor/wymeditor.git
```

2. Install `make` and Java.
3. Download Closure Compiler application, extracting `compiler.jar` into your `wymeditor` directory.
4. Run `make` from your git clone:

```
$ cd wymeditor
$ make min_closure archive
```

2.2.2 Testing WYMeditor

WYMeditor includes a full unit test suite to help us ensure that the editor works great across a variety of browsers. The test suite should pass in any of our supported browsers and if it doesn't, please file a bug so we can fix it!

Running the Tests Locally

Running the Tests in a Browser

1. Get a copy of the source using git:

```
$ git clone git://github.com/wymeditor/wymeditor.git
```

2. Put your source behind some kind of web server (apache, nginx, etc). If you don't have one installed or don't want to fuss with configuration, you can use python's HTTP server:

```
$ cd /path/to/my/wymeditor/src
$ python -m SimpleHTTPServer
```

3. The unit test suite is located at `src/test/unit/index.html`, so if you used the python instructions, open up your browser to http://localhost:8000/test/unit/index.html.

All green means you're good to go.

Running the Tests from the Command Line

In addtion to the browser test suite, you can also run the unit tests from the command line in a headless Phantom.js browser using Grunt by following these instructions:

1. If you haven't already, get a copy of the source using git:

   ```
   git clone git://github.com/wymeditor/wymeditor.git
   ```

2. Use NPM to install the Grunt requirements by running this command in the root directory of the project:

   ```
   $ npm install
   ```

   Note: You might have to run this command as the the root user on your system.

3. Use NPM to install the Grunt CLI.

   ```
   $ npm install -g grunt-cli
   ```

   Note: You might have to run this command as the the root user on your system.

4. Finally, run the tests by running the `test` Grunt task in the root directory of the project:

   ```
   $ grunt test
   ```

If the task runs with no errors or failures, you're good to go.

Testing Different jQuery Versions

The unit tests can be run with the different versions of jQuery hosted on Google's CDN. To do this when running tests in a browser, append the URL parameter `?jquery=<version>` to the test suite URL. To do this when running tests from the command line with Grunt, include the parameter `--jquery=<version>` when running the `test` task.

For a browser example, to test with jQuery 1.8.0 against a local server on port 8000, use the URL: http://localhost:8000/test/unit/index.html?jquery=1.8.0.

For a command line example, to test with jQuery 1.8.0 using Grunt, use the command:

```
grunt test --jquery=1.8.0
```

Grunt Watch Task

Besides just the task for running the tests, the Grunt for the project also includes a `watch` task that can be used to have Grunt automatically run the unit tests anytime a file in the project's source files is modified. This task can be invoked by running this command in the project's root directoy:

```
$ grunt watch
```

Just like with the normal `test` task, a specific version of jQuery can be specified to be used for the tests in the `watch` task by including the parameter `--jquery=<version>` when running the command to start the task.

Travis CI

WYMeditor is set up on Travis CI so that the unit tests are run automatically using the `test` Grunt task with several different versions of jQuery whenever commits are submitted to the Git repositiory for the project. Any submitted pull requests should pass these tests.

2.2.3 WYMeditor Architecture

At a high level, WYMeditor is a jQuery plugin that replaces a textarea with a designMode iframe surrounded by an editor skin and editor controls/buttons. When you call `$('.myEditorClass').wymeditor()`, several things happen:

1. WYMeditor uses your configuration/settings to build out HTML templates for all of the various components.
2. Your textarea is hidden and WYMeditor uses the generated HTML to insert the editor buttons and interface in the same spot.
3. An iframe with `designMode=true` is created and inserted as the area you actually type in.
4. WYMeditor detects your browser and uses a form of prototype inheritance to instantiate the browser-specific version of the editor. The editor is the collection of event listeners, browser-specific DOM manipulations and interface hacks needed to provide the same behavior across IE, Firefox, Chrome, Safari and Opera.

Code Structure

WYMeditor's source is organized in to several modules corresponding to large pieces of functionality and all of the actual code lives in `src/wymeditor/`. The `build/` directory contains build tools and is the default location where bundled/packed/minified/archived versions of the project can be built (to `build/build/`) using the `Makefile`.

Inside `src/` you also have:

- `jquery/` – Includes the bundled version of jQuery and jQuery UI.
- `test/` – Includes both manual and unit tests. The manual tests in the .html files have various editor configurations and combinations of plugins and options.
  - `unit/` – Here lives the automated test suite which makes it possible to make a change without breaking 90 other things. The unit test suite is based on QUnit.

core.js

This is the core of the project including the definition of the WYMeditor namespace, project constants, some generic helpers and the definition of the jQuery plugin function.

The most important object in core is `WYMeditor.editor()` which is the object/function that `jquery.fn.wymeditor()` farms out to for all of the work of actually building the editor. `WYMeditor.editor()` does the basic job of building the `options` object (which for most cases is 99% the defaults), locating your JavaScript files and calling `init()` (which is defined in `editor/base.js` for the heavy lifting.)

parser.js

Here be dragons. `parser.js` is fundamentally responsible for taking in whatever HTML and CSS the browser or user throws at it, parsing it, and then spitting out 100% compliant, semantic xHTML and CSS. This also includes some

work to correct invalid HTML and guess what the user/browser probably actually meant (unclosed li tags, improperly nested lists, etc.) It uses several components to do its job.

1. A lexer powered by parallel regular expression object
2. A base parser and an xHTML-specific parser
3. An XhtmlValidator to drop nonsense tags and attributes
4. A SAX-style listener to clean up the xHTML as it's parsed (this does most of the magic for guessing how to fix broken HTML)
5. A CSS-specific lexer and parser. This is mostly just used to take in CSS configuration options.

editor/

This folder is where most of the magic happens. It includes `base.js` for doing most of the heavy lifting and anything that can be done in a cross-browser manner. It takes the options you passed to the jQuery plugin (defined in `core.js`) and actually creates all of the UI and event listeners that drive the editor.

In the `init()` it also performs browser detection and loads the appropriate browser-specific editor extensions that handle all of the fun browser-specific quirks. The extensions are also located in this folder.

iframe/

This folder contains the HTML and CSS that's used to create the actual editor iframe (which is created by the `init()` method in the `WYMeditor.editor` object defined primarily in `editor/base.js`.) By switching the iframe source configuration, you choose which iframe to use (default is of course, `default`.)

Of special interest is the `wymiframe.css` file inside your chosen iframe (eg. *src/wymeditor/iframe/default/wymiframe.css*.) This file defines the signature blue background with white boxes around block-level elements and with the little "P, H2, CODE" images in the upper left.

skins/

The `skins/` folder is structured like the `iframe/` folder, with folders corresponding to different available skins and a default of `default`. An editor skin allows customization of the editor controls and UI, separate from the editable area that where a user actually types. The skin generally contains CSS, JS and icons and hooks in to the bare HTML that's produced by `WYMeditor.editor` using defined class names that correspond to different controls. The best way to understand and create/edit a skin is to look at the HTML constants defined in `WYMeditor` (inside `core.js`) and compare them to the CSS/JS defined in an existing skin (e.g. `src/wymeditor/skins/default`).

lang/

This is where translations to other languages live. Each file defines a specific `WYMeditor.STRINGS.<language code>` object with mappings from the English constant to the translated word.

plugins/

This is where all plugins bundled with WYMeditor live. There's currently quite a lot of variance between the ways plugins are created an organized, but they're at least organized with a top-level folder in the plugins directory with the plugin's name. In general, the name of the main plugin file has the format `jquery.wymeditor.<plugin_name>.js`.

A set of plugin system hooks is on the roadmap, but for now most plugins modify things in different ways and are relying on APIs that are not guaranteed. 1.0 will provide those guaranteed APIs. See *Plugin System Architecture*.

2.2.4 Expected Cross-browser behavior of the Cursor in Reaction to Keystrokes

For usability, it's very important that the common navigation keys (up/down/enter/backspace) should behave as a the user expects. This means that whenever possible, any cross-browser behavioral differences should be corrected.

General Guidelines

- The Enter and Backspace keys should be compliments with regards to navigation. Doing one and then the other should return to the same state.
- The Up and Down keys should be compliments.
- The Up and Down keys should *not* create new blocks. Only move the cursor between them.
- When navigating to "blue space" (areas without blocks) via the nav keys or the mouse, a paragraph block should be created on first content keystroke.
- Backspace at the start of a block joins the contents of the current block with the contents of the previous.
- Enter when in a block creates a new block after the current, with the content after the cursor in the new block.
- Tables, Images, Blockquotes and Pre areas are "special".

Guide

The | character represents the cursor.

Paragraph at Start

```
<p>|text</p>
```

Up

No Change

Down

No Change

Enter

```
<p></p>
<p>|text</p>
```

backspace

No Change

2.2.5 Selection API

The Selection API will be used to ease the interaction with selected text and the current cursor position. If you want to handle an event yourself (and stop the default one), it's important to know where the cursor currently is. But often you only need to know the position relative to a container element and the container itself. There some problems that arise, such as nested tags. In this document, | will be used to indicate the current cursor position:

```
<p>Some <em>nice|</em> text</p>
```

Of course you don't want returned as container element, but the <p>.

Here's my proposal of some basic functionality. That was all I needed for my prototype. The following code is Javascript pseudocode:

```
selection = {
    // properties

    // the original DOM Selection object
    //(http://developer.mozilla.org/en/docs/DOM:Selection)
    original:

    // the node the selection starts. "start" is the most left position of the
    // selection, not where the user started to select (the user couls also
    // select from right to left)
    startNode:

    // the node the selection ends
    endNode:

    // the offset the cursor/beginning of the selection has to it's parent nodenode
    startOffset:

    // the offset the cursor/end of the selection has to it's parent nodenode
    endOffset:

    // whether the selection is collapsed ot not
    isCollapsed:


    // That one don't need to be implemented at the moment. It's diffcult as
    // one need to define what length is in this context. Is it the number of
    // selected characters? Or the number of nodes? I'd say it's the number of
    // selected characters of the normalized (no whitespaces at end or beginning,
    // multiple inner whitepsaces collapsed to a single one) selected text.
    length:

    // methods

    // Returns true if selection starts at the beginning of a (nested) tag
    // @param String jqexpr The container the check should be performed in
    // @example element = <p><b>|here</b></p>, element.isAtStart("p") would
    // return true
    isAtStart: function(jqexpr)

    // Returns true if selection ends at the end of a (nested) tag
    // @param String jqexpr The container the check should be performed in
    // @example element = <p><b>here|</b></p>, element.isAtEnd("p") would
    // return true
    isAtEnd: function(jqexpr)
```

```
    // set the cursor to the first character (it can be nested),
    // @param String jqexpr The cursor will be set to the first character
    //     of this element
    // @example elements = <p><b>test</b></p>, element.cursorToStart("p")
    //     will set the cursor in front of test: <p><b>|test</b></p>
    cursorToStart: function(jqexpr)

    // set the cursor to the last character (it can be nested),
    // @param String jqexpr The cursor will be set to the last character
    //     of this element
    // @example elements = <p><b>test</b></p>, element.cursorToEnd("p")
    //     will set the cursor behind test: <p><b>test|</b></p>
    cursorToEnd: function(jqexpr)

    // removes the current selection from document tree if the cursor isn't
    // collapsed.
    // NOTE Use the native function from DOM Selection API:
    // http://developer.mozilla.org/en/docs/DOM:Selection:deleteFromDocument
    // NOTE First I had also "deleteFromDocument()" in the Selection API, but
    // it isn't needed as a "sel.deleteIfExpanded()" would do exactly the same
    // as "sel.deleteFromDocument()" with the only difference that this one
    // has a return value
    // @returns true:  if selection was expanded and therefore deleted
    //          false: if selection was already collapsed
    // @example // do what delete key normally does
    //          if (!sel.isCollapsed)
    //          {
    //              sel.original.deleteFromDocument();
    //              return true;
    //          }
    //          will be now:
    //          if (sel.deleteIfExpanded())
    //              return true;
    deleteIfExpanded: function()
};
```

Todo

- save selection (and current cursor position)

2.2.6 Plan for Insertion Functions

Current plan for a clean API around inserting inline and block elements and text.

Deprecate `insert` and alias it to `insertInline`.

`insertInline`

Basically the same as the current `insert` function with a few exceptions.

- Inserts the contents where the current selector is inside the current block element.
- If the selector is in the body, it creates a paragraph to wrap the contents inside, if needed.
- If `insertInline` is called with HTML representing a non-nestable block level element, an exception is raised instead of attempting to guess the proper behavior.

insertBlockAfter

(This is @samuelcole's insert_next, effectively)

Used to insert block-level elements after the currently-selected block-level element.

- If given a string representing a text node, wraps the text in the appropriate container (`p` or `li` tag, depending).
- No matter the current selection, does *not* split the current block.

insertBlock

Used to insert a block-level element and split the current block-level element on the selection boundary.

- If given a string representing a text node, wraps the text in the appropriate container (`p` or `li` tag, depending)

Examples

Inserting `<p>inserted</p>` with `|` representing current cursor.

Middle of Node

```
<p>before| after</p>
<p>before</p><p>inserted</p><p>| after</p>
```

Start of Node

```
<p>|before after</p>
<p>|before after</p><p>inserted</p>
```

End of Node

```
<p>before after|</p>
<p>before after|</p><p>inserted</p>
```

2.2.7 Plugin System Architecture

A proper plugin system should provide one obvious way to perform all of the common things that plugins do. It should still be possible, however, to use plain Javascript to do uncommon things.

Goals

- Plugins should read such that someone who understands Javascript should be able to follow along without knowing all of the details of the plugin API. This means that a little more "boilerplate" is better than magic and explicit is better than implicit. An example of explicit:

```
function TableEditor(name, wymeditor, options) {
    this.name = name;
    this.wymeditor = wymeditor;
    this.options = options;
}

TableEditor.prototype.init = function() {
    this.editor.addButton({'name': 'AddRow', 'title': 'Add Row', 'cssClass': 'wym_tools_addr
}
```

```
TableEditor.prototype.bindEvents = function() {
    var tableEditor = this;
    var wym = this.wymeditor;

    wym.buttons.getButton('AddRow').find().click(function(evt) {
        return tableEditor.handleAddRowClick(evt);
    });
}
```

An implicit way to do things would be to set up some magically-named class attribute that is automatically used by WYMeditor at some point during initialization to create these things.

- Very little method replacing and attribute mangling of the wymeditor object itself should be necessary.
- Most/all of a plugins UI actions (creating dialogs, adding buttons etc) should be done through API calls, allowing editor-wide standardization and theming.
- Plugins should have strong hooks into actions so that they're able to clean up the DOM to fix cross-browser problems.
- All currently-core WYMeditor actions should be migrated to being stand-alone plugins.
- The plugin API shouldn't be responsible for automatically locating Javascript files. The only way to efficiently handle that is on the server side and in the HTML file itself.

Enabling a Plugin

All well-behaving plugins should be explicitly enabled through the `plugins` configuration option on editor initialization. Plugins that alter WYMeditor behavior without being explicitly enabled (like the 0.5.x embed plugin) are considered misbehaving.

The `WYMeditor` `plugins` configuration option is where plugin configuration occurs. This object is an array of objects with the plugin's `name`, `func`, and `options`. For example:

```
jQuery('.wymeditor').wymeditor({
    plugins: [
        {
            name: 'table',
            func: TableEditor,
            options: {enableCellTabbing: false}
        }
    ]
});
```

API

WYMeditor.plugins.addPlugin(<pluginName>, <pluginFunction>, <configurationObject>);

WYMeditor.plugins.getPlugin(<pluginName>);

WYMeditor.buttons.addButton(<options>);

WYMeditor.buttons.getButton(<buttonName>);

WYMeditor.buttons.removeButton(<buttonName>);

WYMeditor.dialogs.createDialog(<dialogName>, <options>, <callback>);

WYMeditor.dialogs.destroyDialog(<dialogName>, <options>, <callback>);

WYMeditor.addXhtmlCleanup(<cleanupName>, <cleanupFunction>);

WYMeditor.removeXhtmlCleanup(<cleanupName>);

WYMeditor.addDomCleanup(<cleanupName>, <cleanupFunction>);

WYMeditor.removeDomCleanup(<cleanupName>);

Note: The following methods already exist:

WYMeditor.editor.findUp(); WYMeditor.editor.container(); WYMeditor.editor.update(); WYMeditor.editor.html(); WYMeditor.editor.xhtml(); WYMeditor.editor.switchTo(); WYMeditor.editor.wrap(); WYMeditor.editor.unwrap(); WYMeditor.editor.setFocusToNode(); WYMeditor.editor.exec(<commandName>);

2.2.8 Dev Process Improvements

This page is where I get to brain-dump ideas for improving the development/testing/packaging/documentation/etc process because making Issues for speculation feels dirty.

Standardize the Build Process on Grunt.js

It's a Javascript project, so let's use Javascript tools. Via grunt, folks should be able to: * Make a build * Run the qunit tests in a headless PhantomJS thing * Run the selenium tests * Check code for style problems

This is started here: https://github.com/wymeditor/wymeditor/tree/issue_360

Documentation

- Convert all of the existing docs from the old wiki to RST for hosting on Read The Docs
- Take a pass to remove all of the obvious errors and new stuff
- Move most of the good info from the readme to a good sphinx structure
- Move the stuff that's unique on the wiki to the sphinx structure

Run the Unit Tests in Every Browser on Every Build

Testling-CI seems like the way to go for running our unit tests across our supported browsers. It won't work for our Selenium tests, but it will at least make it easy to catch regressions and the like when lazy developers *cough*me*cough* don't test in all of the IE's.

Use travis-CI to run Selenium Tests in Chrome

It would be nice to run our Selenium tests in every supported browser on every commit, but it's at least easy to run them using Phantomjs inside travis-ci: http://about.travis-ci.org/docs/user/gui-and-headless-browsers/

Find Something to run Selenium Tests in Other Supported Browsers

There has to be a thing. Find that thing, and then use it.

Find a Better Way of Hosting the Demos

Joyent has discontinued their no.de node PaaS that we previously used to host the demos. Either move to their Nodejitsu replacement, or find something else.

2.3 Using WYMeditor

2.3.1 Introduction

These guidelines will teach you why to use certain techniques when creating content and how to work with them in WYMeditor. Following these will keep the structure and meaning of your content correct - making it more flexible to technologies like SEO1 (Search Engine Optimization), screen readers2, mobile phone technologies, printers etc. Following these guidelines will also make your content accessible to a wider range of people, with or without disabilities

2.3.2 Guidelines

1. Use elements as intended

HTML is a so called markup language. It's purpose is to give the content meaning and structure, not visual effects. Therefore, it's important to use elements as intended to not break meaning, structure and compatibility to other technologies. If you want to style your content you've created with WYMeditor, you shall learn to use CSS (Cascading Style Sheets).

1.1. Headings

Using a logical and proper heading structure is among one of the most important SEO techniques. Always use a correct hierarchy. Start with Heading 1, followed by Heading 2, Heading 3... etc. A proper heading structure will also help users to find what they are looking for, faster.

To create a heading you simply place your marker where you want the heading to go and then select "Containers > Heading 1" for example.

1.2. Tables

Creating lean and accessible tables is a craft but WYMeditor makes it easy. Only use the Table element to arrange different kinds of data - time tables, statistics, member listings etc. Use the Table Header container to mark up headers properly. Doing this wrong will confuse users with screen readers when accessing tables, as headers aren't part of the data.

To create a Table Header you simply place your marker in a table cell and select "Containers > Table Header".

It's also good practice to give users the ability to do a quick overlook of the table's content by filling in a short description in the Caption field. You can do that in the dialog appearing when creating the table.

1.3. Lists

It's important to mark up lists of items the right way, using Ordered list or Unordered list. Doing this wrong makes it harder for users to overlook the content and will also confuse users with screen readers. Creating a list using Shift+Enter to separate items is therefore highly depreciated as both structure and meaning will break in the document.

To create a list you simply use one of the two list buttons in WYMeditor's toolbar. Hit enter twice to "jump out of" a list.

1.4. Indent and outdent

The Indent and Outdent icons are only intended to create nested lists of list items (as described in 1.3). Place the marker on a list item and hit the Indent icon to make it go sub level. To undo this move or to make it go back one level, simply hit the Outdent icon.

1.5. Blockquotes

Mark up your quoted text by using the Blockquote container. Doing this the right way will help search engines and users with screen readers to understand your text and give it relevant meaning.

2. Provide alternatives

2.1. Alternative text

An alternative text string shall be accurate and equivalent in presenting the same content and function as the image. This also helps to identify the relevance and meaning of your image which improves SEO and accessibility for user with screen readers.

You can enter the Alternative text in the image dialog, appearing when adding or editing an image. The only exception to leave the field empty is when your image is for decoration purpose only.

2.2. Title attribute

To provide additional information of a link or and image you can provide a Title text. This helps several other technologies to identify the relevance and meaning of your elements which improves SEO and accessibility for user with screen readers.

You can enter the Title text in the image/link dialog, appearing when adding or editing an element.

3. Write understandable

Writing understandable is among one of the most important checkpoints in creating accessible content. Unfortunately computers can't do anything about this, yet. But with WYMeditor you have got the tools to make it down the right way.

3.1. Descriptive headings

A heading element shall briefly describe the topic of the section it introduces making it easier to overlook both for all users. This will also improve the SEO aspects of your content.

3.2. Descriptive link texts

A link text shall be completely descriptive of its location, which makes it a lot easier to overlook and skim-read the content for all users. This will also improve the SEO aspects and the "link indexing" procedure for users with screen readers. "Click here" links are therefor highly depreciated.

2.3.3 Word References

SEO (Search Engine Optimization) "Search engine optimization" is a conception for everything that makes your content rank higher when using search engines like Google.

Screen reader A screen reader is a text-only web browser that dictates webpages' content to visually disabled peoples. In this text "screen readers" refers to all sorts of assistive and adoptive web technologies.

2.4 Contributed Examples and Cookbooks

2.4.1 Integrate WYMeditor in Django Administration

I have managed to easily integrate WYMeditor into Django's administrative interface.

Here is how I did it...

First I copied the wymeditor to my project's static-served files directory, which in my case had an URL prefix of `/site/media/`

There I put the `wymeditor`, `jquery` and a special file `admin_textarea.js`, that I have written myself consisting of:

```
$(document).ready(function() {
    $('head', document).append('<link rel="stylesheet" type="text/css" media="screen" href="/site/me
    $("textarea").wymeditor({
        updateSelector: "input:submit",
        updateEvent:    "click"
    });
});
```

This file instructs the browser to load an additional WYMeditor's CSS and to convert each `<textarea>` HTML tag into a WYMeditor.

In each of the Django models that I whished to set WYMeditor for, I have added the following `js` setting to the Admin section:

```
class ExampleModel(models.Model):
    text = models.TextField()      # each TextField will have WYM editing enabled
    class Admin:
         js = ('/site/media/jquery.js',
               '/site/media/wymeditor/jquery.wymeditor.js',
               '/site/media/admin_textarea.js')
```

That is it. If you wish to use WYM in your own Django app, just follow the steps and replace the `/site/media/...` with whatever your static media prefix is.

Using the filebrowser Application

To integrate Wymeditor with the Django filebrowser, put the code below in a file, set the `fb_url` variable to point to your filebrowser instance and add the file to your Javascript headers:

```
<script type="text/javascript" src="/media/wymeditor/plugins/jquery.wymeditor.filebrowser.js"></scrip
```

or in your admin.py:

```
class Media:
    js = ('/media/wymeditor/plugins/jquery.wymeditor.filebrowser.js',)
```

Add the `postInitDialog` parameter to the Wymeditor initialization:

```
$('textarea').wymeditor({
  postInitDialog: wymeditor_filebrowser
});
```

If you already have a `postInitDialog` function, you need to put a call to `wymeditor_filebrowser` inside that function:

```
$('textarea').wymeditor({
  postInitDialog: function (wym, wdw) {
      // Your code here...

      // Filebrowser callback
      wymeditor_filebrowser(wym, wdw);
  }
});
```

Then you should be able to click on the Filebrowser link to select an image.

Code:

```
wymeditor_filebrowser = function(wym, wdw) {
  // the URL to the Django filebrowser, depends on your URLconf
  var fb_url = '/admin/filebrowser/';

  var dlg = jQuery(wdw.document.body);
  if (dlg.hasClass('wym_dialog_image')) {
    // this is an image dialog
    dlg.find('.wym_src').css('width', '200px').attr('id', 'filebrowser')
      .after('<a id="fb_link" title="Filebrowser" href="#">Filebrowser</a>');
    dlg.find('fieldset')
      .append('<a id="link_filebrowser"><img id="image_filebrowser" /></a>' +
              '<br /><span id="help_filebrowser"></span>');
    dlg.find('#fb_link')
      .click(function() {
        fb_window = wdw.open(fb_url + '?pop=1', 'filebrowser', 'height=600,width=840,resizable=yes,sc
        fb_window.focus();
        return false;
      });
  }
}
```

2.4.2 Image Gallery Implementation Example

The purpose of this page is to explain how an image gallery built with jQuery's jCarousel plugin and based on data loaded via AJAX can be integrated into WYMEditor. The example is built using CodeIgniter (CI) for structure and queries. It uses a series of JS includes in the image dialog, a CI controller function called via AJAX, and a CI model function containing an active record query on an images database. This example presupposes that you have a functioning install of WYMEditor v0.4 and you are relatively familiar with hacking it. It also assumes that you have the source files for and know how to implement an image gallery based on jCarousel. More information and source downloads for jCarousel can be found here: http://sorgalla.com/jcarousel/

1. Including Plugin Code

You will need to include 3 js files and 2 css files into the dialogs' HTML. This is done by using the `dialogHtml` option, which makes up the outer shell of the dialog boxes that are opened by the top buttons in WYMeditor. You will need to change the paths here to match up with whatever your setup is:

```
$('.wymeditor').wymeditor({

  //options

  dialogHtml: "<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//EN'"
    + " 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>"
```

```
    + "<html><head>"
    + "<link rel='stylesheet' type='text/css' media='screen'"
    + " href='"
    + WYM_CSS_PATH
    + "' />"
    + "<title>"
    + WYM_DIALOG_TITLE
    + "</title>"
    + "<script type='text/javascript'"
    + " src='"
    + WYM_JQUERY_PATH
    + "'></script>"
    + "<script type='text/javascript'"
    + " src='"
    + WYM_BASE_PATH
    + "jquery.wymeditor.js'></script>"
    + "<script type='text/javascript' src='/scripts/jquery.imager.js'></script>"
    + "<script type='text/javascript' src='/scripts/easing.js'></script>"
    + "<script type='text/javascript' src='/scripts/jquery.jcarousel.js'></script>"
    + "<link rel='stylesheet' type='text/css' href='/css/jquery.jcarousel.css' />"
    + "<link rel='stylesheet' type='text/css' href='/css/skins/tango/skin.css' />"
    + "<style type='text/css'>"
    + ".jcarousel-skin-tango.jcarousel-container-horizontal { width: 85%; }"
    + ".jcarousel-skin-tango .jcarousel-clip-horizontal { width: 100%; }"
    + "</style>"
    + "</head>"
    + WYM_DIALOG_BODY
    + "</html>",

  dialogImageHtml:  "<body class='wym_dialog wym_dialog_image'"
    + " onload='WYM_INIT_DIALOG(" + WYM_INDEX + ")'"
    + ">"
    + "<form>"
    + "<fieldset>"
    + "<legend>{Image}</legend>"
    + "<div class='row'>"
    + "<label>{URL}</label>"
    + "<input type='text' class='wym_src' value='' size='40' />"
    + "</div>"
    + "<div class='row'>"
    + "<label>{Alternative_Text}</label>"
    + "<input type='text' class='wym_alt' value='' size='40' />"
    + "</div>"
    + "<div class='row'>"
    + "<label>{Title}</label>"
    + "<input type='text' class='wym_title' value='' size='40' />"
    + "</div>"
    + "<div class='row row-indent'>"
    + "<input class='wym_submit' type='button'"
    + " value='{Submit}' />"
    + "<input class='wym_cancel' type='button'"
    + "value='{Cancel}' />"
    + "</div>"
    + "</fieldset>"
    + "</form>"
    + "<div id='gallery'><h2>Loading images, please wait...</h2></div>"
    + "</body>"
```

```
  //other options

});
```

Note: Options specified when calling a new WYMEditor instance must be separated by commas. If you add in some other options to the example shown above, you will need to add a comma to the end of the dialogImageHtml assignment as well as all the rest of the new options except the last one.

---

jquery.js, easing.js, and jquery.jcarousel.js are all pre-built components that can be downloaded from various sites - see the jCarousel link above. jquery.imager.js is a custom script that will be built below. jquery.jcarousel.css and skins/tango/skin.css are parts of jCarousel, please refer to the jCarousel link above for more information. The header styles are used to make jCarousel expand / contract based on the width of the dialog window.

2. Adding Injection Target to Dialog Body

OK, now all of our scripts should be in place. Upload the file, refresh your WYMEditor install and open the image dialog; you should see all that stuff in the head of the document. It's usually a good idea to copy the paths out of the source here, paste them into another browser window, and make sure they open; this is just to make sure you have the paths correct and all the files are in the right place.

The next step is to add a target `div` to the image dialog body HTML. This gives us a place to inject our dynamic image list a bit later once we have built it with AJAX and PHP. In the `dialogImageHtml` string (around line 619) add the following line:

```
+ "<div id='gallery'><h2>Loading images, please wait...</h2></div>"
```

This line should go after the form but before the close body tag, like so:

```
+ "</div>"
+ "</fieldset>"
+ "</form>"
+ "<div id='gallery'><h2>Loading images, please wait...</h2></div>"
+ "</body>",
```

Once this is in place, upload and refresh WYMEditor again, then re-open the image dialog. You should now see a big fat "Loading images, please wait..." message underneath the form. OK now its time for the slick stuff.

3. AJAX Call from Javascript

Make a new javascript file and save it as jquery.imager.js or whatever else you want to call it; just make sure it's included in the dialog HTML. Paste the following code into the js file:

```
// JavaScript Document

var $j = jQuery.noConflict();

$j(function() {
    //set up your AJAX call
    $.ajax({
      type: "POST",
      url: "http://www.your-domain.com/controller/ajaxer", //path to your PHP function
      data: "img=test&stamp=now",                          //not required for this example,
      success: function(msg){                                      //trigger this code if t
        /*
            The PHP function needs to return an image UL with the following prototype:
```

```
            <ul id="mycarousel" class="jcarousel-skin-tango">
                <li><img src='http://www.test.com/upload/titan.jpg' width='68' height='60' alt='Tenne
                <li><img src='http://www.test.com/upload/canyon.jpg' width='68' height='60' alt='Gra
                <li><img src='http://www.test.com/upload/img_2_big.jpg' width='68' height='60' alt='
            </ul>

            The returning HTML is contained in the msg variable.
        */

        //inject the image list into the target div with ID of "gallery"
        $("div#gallery").html(msg);

         //Once the list is in place we can create a new instance of jCarousel and point it at the in
         //which has an ID of 'mycarousel'. For more information on these options see http://sorgalla
        jQuery('#mycarousel').jcarousel({
            easing: 'backinout',
            visible: 5,
            animation: 500

        });

        //assign behaviors to the jCarousel thumbnails, triggered when they are clicked upon.
        $(".jcarousel-skin-tango img").click(function() {
            //$(this) is a reference to the thumbnail that got clicked
            $("input.wym_src").val($(this).attr('src'));            //inject the thumb's src attribut
            $("input.wym_alt").val($(this).attr('alt'));            //inject the thumb's alt attribut
            $("input.wym_title").val($(this).attr('title'));        //inject the thumb's title attri

            //loop through all the images and remove their "on" states if it exists
            $(".jcarousel-skin-tango img").each(function(i){
              $(this).removeClass("on");
            });
            //add "on" state to the selected image
            $(this).addClass("on").fadeIn('slow');
        })
      }
    });
});
```

Now you have a structure and behaviors for inserting your image code into the dialog. Now all you need is some images!

4. Database Structure

I have an images mySQL table with the following structure:

```
img_id          int(11)
img_upload_date int(11)
img_upload_by   int(11)
img_name        varchar(64)
img_file_name   varchar(64)
img_size        float
img_width       int(11)
img_height      int(11)
img_string      varchar(64)
img_alt         varchar(255)
```

5. PHP / CodeIgniter Functions

Basically you can make an AJAX call to any PHP page that will return a list of the images you want to display in the gallery. It can be connect to a database or not; that's up to you. For my particular setup, I have a CI function in my Content model called `get_images()` that returns either a list of all images in the DB or a specific image if you send a valid ID. The model function goes like this:

```
//ARGS: image ID (int) or -1 to get all images
function get_images($img_id)
{
    if($img_id != -1){ $this->db->where('img_id', $img_id); }
    $this->db->orderby('img_upload_date desc');
    $query = $this->db->get('tq_images');

    if($query->num_rows() > 0)
    {
        return $query->result_array();
    } else {
        return NULL;
    }
}
```

This will return either one or many rows of the database, ordered by the date the image was added, descending. If you send -1 as `$img_id` it will return all images; if you send it a number it will return a specific row if it's a valid ID. If the function can't find any rows based on what you sent it, it will return NULL. If you need more information about codeigniter or model functions see http://codeigniter.com/user_guide/.

6. Put it all Together

Now you will make a controller function that is actually accessible via a browser or AJAX call. Open a CI controller and insert the following function:

```
function ajaxer()
{
    //pull in data from javascript AJAX call (not used for now)
    $img = $_POST['img'];
    $stamp = $_POST['stamp'];

    //call your model function
    $img = $this->Content->get_images(-1);
    //create a the return string. This is the structure for your jCarousel list.
    $lst = "<ul id='mycarousel' class='jcarousel-skin-tango'>\n";
    //loop through the images record set. This should be a list of all the images you want to display
    foreach($img as $i)
    {
        //call a custom function in another model to format the date...you probly don't need this
        $date = $this->Page->get_date($i['img_upload_date']);
        //Build a list item for each image in the database. Insert values as needed.
        //This will produce an unordered list with the prototype specified in jquery.imager.js
        $lst .= "<li><img src='". base_url() . "upload/" . $i['img_file_name'] ."' width='68' height=

    }
    //close the list
    $lst .= "</ul>\n";
    //return the list to the dialog
    echo $lst;
}
```

And that's pretty much it. When you are setting up your AJAX call, its URL attribute should be pointed at this controller function. `ajaxer()` will call the model function outlined above then process the returned recordset into an HTML list. Echoing out the list will return it to your Javascript code as the `msg` variable I mentioned above. You should already have code in place in jquery.imager.js to handle the incoming data and inject it where it needs to go. In that same script you have already specified click behaviors for each thumbnail in the list. If you need more information about codeigniter or controller functions see http://codeigniter.com/user_guide/.

7. Using it

Now upload everything and refresh WYMEditor. If you set everything up correctly, you should see your jCarousel load in underneath the dialog form. If you click on an image, you should see its values pop into the input boxes above the carousel. Once you have the correct information in the boxes, hit "submit" on the dialog as usual and WYMEditor should plop it into your piece. Repeat as needed.

If you have any questions about specific technologies used above, see their respective websites. If you have questions about my code, hit me up at rhinocerous at gmail dot com.

2.4.3 Integrate WYMeditor into sNews CMS

This page explains step by step how to integrate WYMeditor version 0.4 into sNews version 1.6. sNews is a light and open source Content Management System. All information about sNews is available on its official site.

Since sNews already includes an editor, some hacks are mandatory in order to replace the default editor with WYMeditor. These hacks are all listed below. Fortunately, in order to avoid you these tedious code modifications, a ready-to-run archive file has been prepared. Unarchive this file and install sNews according to readme.html before to run it. Hacks made inside sNews have all been enclosed between tags [WYMeditor Hack] for allowing you to easily retreive them.

Install sNews

Download archive file sNews16.zip

Unarchive sNews16.zip under your Web space and follow the install instructions contained into readme.html.

Install WYMeditor

Download archive file wymeditor-0.4.tar.gz

Unarchive wymeditor-0.4.tar.gz into a temporary directory and move directories 'jquery' and 'wymeditor' into 'sNews' root directory

Integrate WYMeditor

All the following code modifications must be made in **snews.php**.

Replace line 325

```
if ($_SESSION[db('website').'Logged_In'] == token()) {js();}
```

with

```
    if ($_SESSION[db('website').'Logged_In'] == token()) {js();
        echo
'   <link rel="stylesheet" type="text/css" media="screen" href="wymeditor/skins/default/screen.css" /
    <script type="text/javascript" src="jquery/jquery.js"></script>
```

```
    <script type="text/javascript" src="wymeditor/jquery.wymeditor.js"></script>
    <script type="text/javascript">
        jQuery(function() {
            jQuery(".wymeditor").wymeditor({
                cssPath: "'.db('website').'wymeditor/skins/default/screen.css",
                jQueryPath: "'.db('website').'jquery/jquery.js",
                wymPath: "'.db('website').'wymeditor/jquery.wymeditor.js"
            });
        });
    </script>';
    }
```

This code loads WYMeditor with its default skin and initializes WYMeditor with correct paths.

Replace line 1048

```
case 'textarea': $output = '<p>'.$lbl.':<br /><textarea name="'.$name.'" rows="'.$rows.'" cols="'.$co
```

with

```
case 'textarea':
    if ($_SESSION[db('website').'Logged_In'] == token()) {
        $output = '<p>'.$lbl.':<br /><textarea class="wymeditor" name="'.$name.'" rows="'.$rows.'" co
    } else {
        $output = '<p>'.$lbl.':<br /><textarea name="'.$name.'" rows="'.$rows.'" cols="'.$cols.'"'.$a
    }
    break;
```

This code allows WYMeditor to replace the 'textarea' block with a WYMeditor instance.

Comment from line 1312

```
/*  echo '<p>';
```

to line 1322

```
echo '</p>';*/[/code]
```

This code disables the default editor toolbar.

Replace line 1327

```
echo html_input('fieldset', '', '', '', '', '', '', '', '', '', '', '', '', '', '<a title="'.l('custo
```

with

```
if ($_SESSION[db('website').'Logged_In'] == token()) {
    echo html_input('fieldset', '', '', '', '', '', '', '', '', '', '', '', '', '', '<a title="'.l('c
} else {
    echo html_input('fieldset', '', '', '', '', '', '', '', '', '', '', '', '', '', '<a title="'.l('c
}
```

This code allows the edited article to be correctly previewed by sNews each time the sNews preview button is clicked.

Replace line 1393

```
echo html_input('submit', $frm_task, $frm_task, $frm_submit, '', 'button', '', '', '', '', '', '', ''
```

with

```
echo html_input('submit', $frm_task, $frm_task, $frm_submit, '', 'wymupdate', '', '', '', '', '', '',
```

This code allows the edited article to be correctly saved when the sNews save button is clicked.

That's all, sNews should now run WYMeditor instead of its default editor. If you experiment problems in running or using WYMeditor inside sNews, you are invited to read this topic on the WYMeditor forum or this one on the sNews forum. If you don't find the answer to your problem, feel free to post a new message.

2.4.4 Integrate WYMeditor into Rails

WYM Editor Helper is a plugin that makes it dead easy to incorporate the WYM editor into your Rails views.

Follow these steps to use:

1. From your project's root, run:

```
$ ruby script/plugin install svn://zuurstof.openminds.be/home/kaizer/svn/rails_stuff/plugins
$ rake wym:install
```

2. Put `<%= wym_editor_initialize %>` in the view that will host the text editing form. Prefereably this goes into your html's HEAD, to keep our html W3C valid. Use `<% content_for :head do %> <%= wym_editor_initialize %> <% end %>` in the view that needs the editor, and `<%= yield :head %>` in the layout. This means the editor will only load when it is truly called for.
3. In your form, instead of i.e. `<%= text_area :article, :content %>`, use `<%= wym_editor :article, :content %>` OR add a `wymeditor` class to the `textarea`.
4. Add a `wymupdate` class to the submit button.

Extra Info

This plugin uses an `svn:external` to automatically get the latest version of WYMeditor. If for some reason the checked out version is not working, you can install a different version like so:

```
$ svn export svn://svn.wymeditor.org/wymeditor/tags/0.4 vendor/plugins/wym_editor_helper/assets/wymed
```

To see what versions are available, check the repository browser at http://files.wymeditor.org/wymeditor/. Keep in mind that if you check out a newer version of WYM, you need to re-run the `wym:install` rake command to actually copy the wym files to the public dir. If you do so, be sure to first back up your configuration file (/javascripts/boot_wym.js) if you made any changes to it.

Updates on this plugin will appear on http://www.gorilla-webdesign.be/artikel/42-WYM+on+Rails

CHAPTER 3

Version 2.0

3.1 Roadmap for WYMeditor 2.0 (Draft)

Version 2.0 of WYMeditor will be a complete rewrite fixing a lot of the issues with the current stable version.

Version 2.0 can be grouped in to four major components: the Selection API or SAPI, the Editor core, the HTML, DOM and CSS parsers and the UI. These four components also make up the four steps we'll need to complete before we release version 2.0. Of course these steps overlap somewhat, and it's possible to work on several of these things in parallel.

3.1.1 General Goals

- Separating the different areas of WYMeditor from each other, making it more modular
- Implementing a solid event system
- Documenting the WYMeditor source more thoroughly
- A more consistent source code (through the *Coding Standard*)

3.1.2 The Roadmap

Step 1: The HTML, DOM and CSS Parsers

In the current stable version there's a HTML parser and a CSS parser. In 2.0, we'll also be introducing a DOM parser, as we can not rely on the innerHtml property due to a couple of issues and lack of flexibility.

Goals

- Document the HTML and CSS parsers
- Get rid of the innerHtml dependency

New Features

- Inline controls
- Place holders for forms/flash/dynamic content

Issues to Fix

- innerHtml: Problems with links and urls (#69)
- innerHtml: Insertion of script elements doesn't work

Resources

- http://ejohn.org/blog/pure-javascript-html-parser/
- http://www.stevetucker.co.uk/page-innerxhtml.php

Step 2: the Selection API

The SAPI in the current stable version is rather incomplete. To make things easier, the excellent IERange library will be used for Internet Explorer compatibility.

The SAPI could also be released as a standalone jQuery plug-in, as it could be useful to a lot of other people and projects.

Goals

- Define a general jQuery plugin for selections and ranges, and integrate the plugin with WYMeditor
- Creating a consistent API that can be used for manipulating content

New Features

- Save and restore selections, allowing support for modal dialogues and the like
- A more solid API

Resources

- http://code.google.com/p/ierange/
- https://developer.mozilla.org/en/DOM/Selection
- https://developer.mozilla.org/en/DOM:range

Step 3: The Editor Core

In 2.0 we'll be moving away from designMode in favour of contentEditable. This will not only reduce the complexity a lot, it will also open up a lot of new interesting possibilities. Through easier interaction with external scripts things such as drag an drop, placeholders and the like are a lot easier to achieve.

Goals

- Move away from designMode in favour of contentEditable
- Solid event handling
- New features
- onChange/isDirty functionality (#138)

Issues to Fix

- Support for different block level elements (#152, #178)

Step 4: The UI

The goal is to create a new clean looking, extendable and skinnable UI that's completely separated the WYMeditor core. This would allow the developer to completely replace the default UI with another one, opening up a lot of different integration possibilities.

Goals

- Create a new clean looking and consistent UI
- Separate the UI from the editor core
- Make it extendable and skinnable
- Make it more user friendly through on-screen help and well thought out layout, features and (visual) feedback

New features

- One toolbar for several editor instances (#97)
- Modal dialogues (#63)

Issues to Fix

- Link editing (#69)
- Block level elements inside list items (#135)

3.2 Contributing

3.2.1 We <3 Contributions

We love your contributions. Anything, whitespace cleanup, spelling corrections, translations, jslint cleanup, etc is very welcome.

The general idea is that you fork WYMeditor, make your changes in a branch, add appropriate unit tests, hack until you're done, make sure the tests still pass, and then send a pull request. If you have questions on how to do any of this, please stop by #wymeditor on freenode IRC and ask. We're happy to help!

3.2.2 Example Process

1. Fork wymeditor to your personal GitHub account.
2. Clone it (`git clone <your personal repo url>`) and add the official repo as a remote so that you easily can keep up new changes (`git remote add upstream https://github.com/wymeditor/wymeditor.git`).
3. Create a new branch and check it out (`git checkout -b my-cool-new-feature`).
4. Make your changes, making sure to follow the *Coding Standard*. If possible, also include a unit test in src/test/unit/test.js.
5. Add the changed files to your staging area (`git add <modified files>`) and commit your changes with a meaningful message (`git commit -m "Describe your changes"`). Make sure and follow the *Coding Standard*.
6. Repeat steps 4-5 until you're done.
7. Add yourself to the AUTHORS file!

8. Make sure unit tests pass in as many browsers as you can. If you don't have access to some of the supported browsers, be sure and note that in your pull request message so we can test them.
9. Make sure your code is up to date (see below) and if everything is fine push your changes to GitHub (`git push origin <your branch>`) and send a *Pull Request*.

3.2.3 Staying up to Date

If your fork or local branch falls behind the official upstream repository please do a `git fetch` and then `merge` or `rebase` to make sure your changes will apply cleanly – otherwise your pull request will not be accepted.

See the GitHub help section for further details.

3.3 Proposed Build Stack

3.3.1 100% Javascript

For WYMeditor 1.0, we're moving to an all-Javascript build stack from the Node.js community. The rationale behind this is simple – use tools that make senese to JS developers. Any person with node.js and npm installed should be able to run a few npm commands and then do development or run a build.

The goal will be to combine asset-management best practices like concatenation and minification for CDNs with automatic file generation for easy development. Editors like TinyMCE are often criticized for poor HTTP performance and there are posts strewn across the interwebs from people attempting to hack together solutions to this problem. We should solve it for everyone by default and even make it easy for folks to make custom builds with just the plugins that they require.

3.3.2 Components

General build tool

There are always miscellaneous tasks and helpers that developers need, from building documentation to bumping version numbers and tagging a new release. It's always nice for new users if there is a consistent way of performing these actions.

Jake

A port of Rake. We should be able to lint, run tests, build documentation, create a full build or create customized builds all from jake.

Asset Management

We ultimately want the ability to produce a trio of one minified javascript file, one minified CSS file and one image file with all requirements. This will include the editor, its skin and any plugins the developer wants to enable (including 3rd-party plugins) along with all of their assets.

Either node-ams or local-cdn. Both provide a file-watching development server via node.js to allow easy development. Both let you statically define and then generate files on demand for releases.

Documentation

What we really need is Sphinx ported to Javascript, but until something like that emerges in the node community, the standard solution is inline documentation plus stand-alone statically-generated HTML documentation.

Inline Documentation

Docco

Allows writing inline docs in pure Markdown. Not as restricting as most other solutions since it isn't modeled after programming paradigms foreign to JavaScript. Also good for consistency across different platforms (GitHub wiki, issues and comments, the new forum, etc.)

Full Documentation

Inline documentation doesn't cover tutorials, API documentation and reference docs. The Django documentation is a good example of what we're going for. For now, it seems most projects are rolling their own combination of statically-generated HTML sites powered by MarkDown. To start, rip off the documentation from a project like express and get started.

Testing

- QUnit for the test framework.
- phantomJS and qunit-tap to run the tests from the command line for quick webkit tests.
- Jenkins for CI to automatically package builds, run static analysis, run phantomJS tests and then use Sauce On Demand to spin up supported browsers for cross-browser testing.
- TestSwarm for coordinating cross-browser tests in all supported browsers.
- node-jshint for static analysis and linting.

CHAPTER 4

Plugins

4.1 Included Plugins with Download

4.1.1 Embed

Add description.

4.1.2 Fullscreen

Add description.

4.1.3 Hovertools

This plugin improves visual feedback by:

- displaying a tool's title in the status bar while the mouse hovers over it.
- changing background color of elements which match a condition, i.e. on which a class can be applied.

4.1.4 List

Add description.

4.1.5 RDFA

Add description.

4.1.6 Resizable

Add description.

4.1.7 Table

Add description.

4.1.8 Tidy

Add description.

4.1.9 Structured Headings

This plugin modifies the styling and function of headings in the editor to make them easier to use to structure documents. Currently in development.

Features

- Simplifies the process of adding headings to a document. Instead of having to choose between a heading 1-6 in the containers panel, those options are replaced with one single "Heading" option that inserts a heading at the same level as the preceding heading in the document.
- Makes it easier to adjust heading levels properly. In order to change the level of a heading, the indent and outdent tools can be used to lower and raise the level of the heading respectively. The indent tool will also prevent a user from indenting a heading more than one level below its preceding heading.
- Automatically numbers headings to better outline a document.
- Provides an optional tool that can be used to automatically fix improper heading structure in a document (although it's still in development).
- More features to come on further development.

Applying Structured Headings Styling outside of the Editor

The numbering added to headings in the editor is not parsed with the content of the document, but rather, it is added as styling using CSS in IE8+, Chrome, and Firefox, or it is added as separate content using a stand-alone JavaScript function with additional CSS styling in IE7.

To apply the heading numbering to a document outside of the editor, follow these steps:

1. Get the CSS for the structured headings. If you are using IE8+, Chrome, or Firefox, enter the `WYMeditor.printStructuredHeadingsCSS()` command in the browser console on the page of the editor using the structured headings plugin to print the CSS to the console. If you are using IE7, it uses different CSS than the other browsers, and you can get this CSS from its stylesheet `structured_headings_ie7_user.css` available in the plugin's directory.
2. Either copy this CSS to an existing stylesheet or make a new stylesheet with this CSS.
3. Apply the stylesheet with the CSS to all of the pages that contain documents that had heading numbering added to them in the editor.
4. If you are using IE7, in addition to the previous steps, add the script `jquery.wymeditor.structured_headings.js` available in the plugin's directory to all of the pages that contain documents that had heading numbering added to them in the editor. This script provides the function `numberHeadingsIE7()` which detects which headings on the page were assigned numbering in WYMeditor and adds the numbering for these headings into the page. This function should be called in a script

on document ready after the `jquery.wymeditor.structured_headings.js` script is included on the page. Here is an example of what that script should look like:

```
jQuery(document).ready(function() { numberHeadingsIE7(); });
```

Browser Support

The plugin supports IE7+, Chrome, and Firefox, but it works less conveniently in IE7 as you can see by the additional steps it requires for implementation in that browser.

4.2 Third Party Plugins

4.2.1 Modal Dialog (by samuelcole)

https://github.com/samuelcole/modal_dialog

Replaces the default dialog behavior (new window) with a modal dialog. Known bug in IE, more information here.

4.2.2 Alignment (by Patabugen)

https://bitbucket.org/Patabugen/wymeditor-plugins/src

Set Text Alignment with classes.

4.2.3 Site Links (by Patabugen)

https://bitbucket.org/Patabugen/wymeditor-plugins/src

A plugin to add a dropdown of links to the Links dialog, especially for making it easier to link to your own site (or any other predefined set).

Can also add a File Upload form to let you upload files right from the Link dialog.

4.2.4 Image Float (by Patabugen)

https://bitbucket.org/Patabugen/wymeditor-plugins/src

Float images with classes.

4.2.5 Image Upload (by Patabugen)

https://bitbucket.org/Patabugen/wymeditor-plugins/src

Adds an Image Upload form to the Insert Image dialog.

4.2.6 Catch Paste

Force automatic "Paste From Word" usage so that all pasted content is properly cleaned.

http://forum.wymeditor.org/forum/viewtopic.php?f=2&t=676