A WYSIWYG Framework

by

Johannes-Lukas Bombach

Submitted to the Fachbereich Informatik, Kommunikation und Wirtschaft
in partial fulfillment of the requirements for the degree of

Master of Science

at the

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT BERLIN

August 2015

Author .....
Johannes-Lukas Bombach
August 26, 2015

Certified by .....
Prof. Dr. Debora Weber-Wulff
Associate Professor
Thesis Supervisor

A WYSIWYG Framework

by

Johannes-Lukas Bombach

Submitted to the Fachbereich Informatik, Kommunikation und Wirtschaft
on August 26, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Browsers do not offer native elements that allow for rich-text editing. There are third-party libraries that emulate these elements by utilizing the `contenteditable`-attribute. However, the API enabled by `contenteditable` is limited and unstable. Bugs and unwanted behavior can only be worked around and not fixed. The library "Type" demonstrates that rich-text editing can be achieved without requiring the `contenteditable` attribute, thus solving many problems contemporary third-party rich-text editor libraries have.

Thesis Supervisor: Prof. Dr. Debora Weber-Wulff
Title: Associate Professor

Acknowledgments

I would like to extend my thanks to my supervisor Prof. Dr. Debora Weber-Wulff for giving me the opportunity to work on a topic I have been passionate about for years.

I would like to thank Marijn Haverbeke for his work on CodeMirror, from which I could learn a lot.

I would like to thank my father for supporting me. Always.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Rich-text editors are commonly used by many on a daily basis. Often, this happens knowingly, for instance in an office suite, when users wilfully format text. But often, rich-text editors are being used without notice. For instance when writing e-mails, entering a URL inserts a link automatically in many popular e-mail-applications. Also, many applications, like note-taking apps, offer rich-text capabilites that go unnoticed. Many users do not know the difference between rich-text and plain-text writing. Rich-text editing has become a de-facto standard, that to many users is *just there*. Even many developers do not realise that formatting text is a feature that needs special implementation, much more complex than plain-text editing.

While there are APIs for creating rich-text input controls in many desktop programming environments, web-browsers do not offer native rich-text inputs. However, third-party JavaScript libraries fill the gap and enable developers to include rich-text editors in web-based projects.

The libraries available still have downsides. Most importantly, only a few of them work. As a web-developer, the best choices are either to use CKEditor or TinyMCE. Most other editors are prone to bugs and unwanted behaviour. Piotrek Koszuliński, core developer of CKEditor comments this on StackOverflow as follows:

> *Don't write wysiwyg editor[sic] - use one that exists. It's going to consume all your time and still your editor will be buggy. We and guys from other... two main editors (guess why only three exist) are working on this for years and we still have full bugs lists ;).*[1]

A lot of the bugs CKEditor and other editors are facing are due to the fact that they rely on so-called "HTML Editing APIs" that have been implemented in browsers for years, but only been standardized with HTML5. Still, to this present day, the implementations are prone to noumerous bugs and behave inconsitently across different browsers. And even though these APIs are the de-facto standard for implementing rich-text editing, with their introduction in Internet Explorer 5.5, it has never been stated they have been created to be used as such.

It's a fact, that especially on older browsers, rich-text editors have to cope with bugs and inconsistencies, that can only be worked around, but not fixed, as they are native to the browser. On the upside, these APIs offer a high-level API to call so-called "commands" to format the current text-selection.

However, calling commands will only manipulate the document's DOM tree, in order to format the text. This can also be achieved without using editing APIs, effectively avoiding unfixable bugs and enabling a consistend behaviour across all browsers.

Furthermore CKEditor, TinyMCE and most other libraries are shipped as user interface components. While being customizable, they tend to be invasive to web-projects.

This thesis demonstrates a way to enable rich-text editing in the browser without requiring HTML Editing APIs, provided as a GUI-less software library. This enables web-developers to implement rich-text editors specific to the requirements of their web-projects.

[1] http://stackoverflow.com/questions/11240602/paste-as-plain-text-contenteditable-div-textarea-word-excel/11290082112900082, last checked on 07/13/2015

1.2 Structure

The first part of this thesis explains rich-text editing on desktop PCs. The second part explains how rich-text editors are currently being implemented in a browser-environment and the major technical differences to the desktop. Part three will cover the downsides and the problems that arise with the current techniques used. Part four will explain how rich-text editing can be implemented on the web bypassing these problems. Part five dives into the possibilities of web-based rich-text editing in particular when using the techniques explained in this thesis.

Chapter 2

Text editing in desktop environments

2.1 Basics of plain-text editing

caret selection input

2.2 Basics of rich-text editing

document tree formatting algorithms

2.3 Libraries for desktop environments

It is no longer needed to implement basic rich-text editing components from the ground up. Rich-text editing has become a standard and most modern Frameworks, system APIs or GUI libraries come with built-in capabilites. Table 2.1 lists rich-text text components for popular languages and frameworks.

Environment	Component
Java (Swing)	JTextPane / JEditorPane
MFC	CRichEditCtrl
Windows Forms / .NET	RichTextBox
Cocoa	NSTextView
Python	Tkinter Text
Qt	QTextDocument

Table 2.1: Rich-text components in desktop environments

Chapter 3

Text editing in browser environments

3.1 Plain-text inputs

Browsers have native plain-text inputs However there are no components for rich text editing. Confining the W3C specifications, there are 2 elements made for text input * input fields and * textareas

3.2 Rich-text editing

3.2.1 Third-party JavaScript libraries

The W3C's HTML5 and HTML5.1 specifications as well as the WHATWG HTML specification do not recommend native components for rich text editing.

Rich-text editing in browsers is only possible though JavaScript. Essentially, libraries enabling rich-text editing display a nested webpage through an iFrame and let the user modify its contents to emulate a rich-text input. Commonly, modification is realized though the browser's so-called "HTML Editing APIs", which will be discussed in Section XXX (HTML Editing APIs).

Attribute	Type	Can be set to	Possible values
designMode	IDL attribute	Document	"on", "off"
contentEditable	IDL attribute	Specific HTMLElements	boolean, "true", "false", "inherit"
contenteditable	content attribute	Specific HTMLElements	empty string, "true", "false"

Table 3.1: Editing API attributes

3.2.2 HTML Editing APIs

In July 2000, with the release of Internet Explorer 5.5, Microsoft introduced the IDL attributes `contentEditable` and `designMode` along with the content attribute `contenteditable`[1][2]. These attributes were not part of the W3C's HTML 4.01 specifications[3] or the ISO/IEC 15445:2000[4], the defining standards of that time.

By setting `contenteditable` or `contentEditable` to "true" or `designMode` to "on", Internet Explorer 5.5 switches the affected elements and their children to an editing mode. This mode makes it possible to

1. Let the user interactively click on and type inside text elements

2. Execute "commands" via JScript and JavaScript

Fetching user inputs (clicking on elements, accepting keyboard input and modifying text nodes) is handled entirely by the browser. No further scripting is necessary other than setting the mentioned attributes on elements. This behavior is inherited by child elements.

In editing mode, calling the method `document.execCommand` will format the currently selected text. Calling `document.execCommand('bold', false, null)` will wrap the currently selected text in `<b>` tags. `document.execCommand('createLink', false, 'http://google.com/')` will wrap the selected text in a link to google.com. However, this command will be ignored, if the current selection is not contained by an element in editing mode.

[1] https://msdn.microsoft.com/en-us/library/ms533720(v=vs.85).aspx, last checked on 07/10/2015

[2] https://msdn.microsoft.com/en-us/library/ms537837(VS.85).aspx, last checked on 07/10/2015

[3] http://www.w3.org/TR/html401/, last checked on 07/14/2015

[4] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=27688, last checked on 07/14/2015

While `designMode` can only be applied to the entire document, `contentEditable` and `contenteditable` attributes can be applied to a subset of HTML elements as described on Microsoft's Developer Network (MSDN) online documentation[5].

With the release of Internet Explorer 5.5 and the introduction of editing capabilities, Microsoft released a sparse documentation[6] describing only the availability and the before-mentioned element restrictions of these attributes.

According to Mark Pilgrim, author of the "Dive into" book series and contributor to the the Web Hypertext Application Technology Working Group (WHATWG), Microsoft did not state a specific purpose for its editing API, but, its first use-case has been rich-text editing[7].

In March 2003, the Mozilla Foundation introduced an implementation of Microsoft's designMode, named Midas, for their release of Mozilla 1.3. Mozilla names this "rich-text editing support" on the Mozilla Developer Network (MDN)[8]. In June 2008, Mozilla added support for contentEditable IDL and contenteditable content attributes with Firefox 3.

Mozilla's editing API resembles the API implemented for Internet Explorer, however, there are still differences (compare [9][10]). Most notably, Microsoft and Mozilla differ in the commands provided to pass to document.execCommand[11][12] and the markup generated by invoking commands[13]. In fact, Mozilla only provides commands dedicated to text editing while Microsoft offers a way to access lower-level browser components (like the browser's cache) using execCommand. This may show, that even though rich-text editing was its first use case and Mozilla implemented it

[5]https://msdn.microsoft.com/en-us/library/ms537837(VS.85).aspx, last checked on 07/10/2015
[6]https://msdn.microsoft.com/en-us/library/ms537837(VS.85).aspx, last checked on 07/10/2015
[7]https://blog.whatwg.org/the-road-to-html-5-contenteditable, last checked on 07/10/2015
[8]https://developer.mozilla.org/en/docs/Rich-Text_Editing_in_Mozilla, last checked on 07/10/2015
[9]https://msdn.microsoft.com/en-us/library/hh772123(v=vs.85).aspx, last checked on 07/10/2015
[10]https://developer.mozilla.org/en-US/docs/Midas, last checked on 07/10/2015
[11]https://developer.mozilla.org/en-US/docs/Midas, last checked on 07/10/2015
[12]https://msdn.microsoft.com/en-us/library/ms533049(v=vs.85).aspx, last checked on 07/10/2015
[13]https://developer.mozilla.org/en/docs/Rich-Text_Editing_in_MozillaInternet_Explorer_Differences, last checked on 07/10/2015

naming it that, this editing API was not originally intended to be used as such.

In March 2008, Apple released Safari 3.1[14] including full support for contentEditable and designMode[15], followed by Opera Software in June 2006[16] providing full support in Opera 9[17]. MDN lists full support in Google Chrome since version 4[18], released in January 2010[19].

Around the year 2003[*MeineTabelle*] the first JavaScript libraries emerged that made use of Microsoft's and Mozilla's editing mode to offer rich-text editing in the browser. Usually these libraries were released as user interface components (text fields) with inherent rich-text functionality and were only partly customizable.

In May 2003 and March 2004 versions 1.0 of "FCKEditor" and "TinyMCE" have been released as open source projects. These projects are still being maintained and remain among the most popular choices for incorporating rich-text editing in web-based projects. *// Technik, wie diese Editoren funktionieren erklÃ€ren.*

Seeing the history of editing APIs, it is understandable how this has become the standard for rich-text editing. However, with its introduction in Internet Explorer 5.5, it has not been stated that the `designMode` end `contentEditable` attributes have been intended to enable rich-text editing. Sections X and Y will discuss the advantages and disadvantages of these APIs.

3.2.3 Advantages of HTML Editing APIs

Higlevel API Wenig Aufwand discussion of WHATWG

[14]https://www.apple.com/pr/library/2008/03/18Apple-Releases-Safari-3-1.html, last checked on 07/10/2015
[15]http://caniuse.com/feat=contenteditable, last checked on 07/10/2015
[16]http://www.opera.com/docs/changelogs/windows/, last checked on 07/10/2015
[17]http://www.opera.com/docs/changelogs/windows/900/, last checked on 07/10/2015
[18]https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Content_Editable, last checked on 07/10/2015
[19]http://googlechromereleases.blogspot.de/2010/01/stable-channel-update_25.html, last checked on 07/10/2015

3.2.4 Disadvantages of HTML Editing APIs

bugs limitations No specifications on what markup to generate (mozilla != ie, mdn has links for that)

3.2.5 Usage of HTML Editing APIs

How Js Libraries work. Maybe how only a few work. StackOverlfow quote, buglists, Medium post, other stuff to find.

3.2.6 DOM manipulation without Editing APIs

In October 1998 the World Wide Web Consortium (W3C) published the "Document Object Model (DOM) Level 1 Specification". This specification includes an API on how to alter DOM nodes and the document's tree[20]. It provided a standardized way for changing a website's contents. With the implementations of Netscape's JavaScript and Microsoft's JScript this API has been made accessible to web developers.

[20]http://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html, last checked on 07/10/2015

Chapter 4

Implementation

4.1 JavaScript library development

No IDEs, tools, not even conventions.

Not for building: Big JS libraries all do it differently. Top 3 client side JavaScript repositories (stars) on github https://github.com/search?l=JavaScript&q=stars%3A%3E1&s=stars&ty Angular.js: Grunt d3 Makefile, also ein custom build script welche node packages aufruft jQuery custom scripts, mit grunt und regex und so

Not for Architecture Angular custom module system with own conventions d3 mit nested objects (assoc arrays) und funktionen jQuery mit .fn ACE mit Klassen, daraus habe ich gelernt

4.2 Ich habe verwendet

Gulp requireJs AMDClean Uglify

JSLint - Douglas Crockford coding dogmatas / conventions JSCS - JavaScript style guide checker

Livereload PhantomJs Mocha Chai

Durch Require und AMDClean schÃűne arbeitsweise (am ende Ãijber bord geworfen) und kleine DateigrÃűÃ§e, wenig overhead.

Automatisierte Client side Tests mit PhantomJs und Mocha/Chai

4.3 Coding conventions

Habe mich grÃűÃ§tenteils an Crockfordstyle orientiert, aber die Klassen anders geschrieben. Habe den Stil von ACE editor verwendet, denn der ist gut lesbar. Lesbarkeit war mir wichtiger als Crockford style. FÃijr private Eigenschaften und Methoden habe ich die prefix convention verwedendet. https://developer.mozilla.org/en-US/Add-ons/SDK/Guides/Contributor_s_Guide/Private_Properties Sie bewirkt keine echte accessibility restriction, aber es ist eine allgemein anerkannten convention und ist auch viel besser lesbar.

4.4 Coding Klassen

Ich habe mich fÃijr Klassen entschieden. Das hat folgende Vorteile:

* Klassen sind ein bewÃďhrtes Konzept um Code zu kapseln, logisch zu strukturieren und lesbar zu verwalten * Durch prototypische Vererbung existiert die funktionalitÃďt von Klassen nur 1x im Browser 7 RAM * Zudem gibt es Instanzvariablen, die fÃijr jede Type Instanz extra existieren und so mehrere Instanzen erlauben * Die instanzvariablem sind meistens nur Pointer auf Instanzen anderer Klassen * Das ganze ist dadurch sehr schlank

4.5 Programmstruktur

Es gibt ein Basisobjekt, das ist die Type "Klasse". Darin werden dann die anderen Klassen geschrieben "Type.Caret", "Type.Selection", "Type.Range", ... Das hat den Vorteil dass das ganze ge-name-spaced ist, so dass ich keine Konflikte mit Systemnamen habe (Range) (und auch nicht mit anderen Bibliotheken) Effektiv gibt es eine (flache) Baumstruktur und so mit Ordnung. FÃijr bestimmte Klassen, "Type.Event.Input", "Type.Input.Filter.X" geht es tiefer. Der zweite Grund ist, dass ich somit alle Klassen die ich geschrieben habe fÃijr Entwickler sichtbar bereit stelle und nicht implizit und versteckt Ãijber irgend nen Quatsch.

UrsprÃijnglich ein MVC konzept geplant mit einem Document Model und verschiedenen Renderern, aber Ãijber den haufen geworfen.

Ich werde jetzt die einzelnen Module erlÃd'ren

4.6 Type

Die Type

4.7 Caret

caret

4.8 Range

range

4.9 Selection

selection

4.10 Selection Overlay

overlay

4.11 Input

input

4.12 Formatting

formatting

4.13 Change Listener

change

4.14 Contents

contents

4.15 Development

developmnent

4.16 Dom Utilities

dom util

4.17 Dom Walker

dom walker

4.18 Environment

env

4.19 Core Api

core api

4.20 Event Api

ev api

4.21 Events

Input input event only event required so far

4.22 Input Pipeline

pipeline ideas

Caret caret

Command command

Headlines head lines

Line Breaks line breaks

Lists lists

Remove remove

Spaces spaces

4.23 Plugin Api

plugin api

4.24 Settings

settings

4.25 Text Walker

text walker

4.26 Utilities

util

Appendix A

Tables

Appendix B

Figures

Bibliography