

A WYSIWYG FRAMEWORK

MASTER'S THESIS

LUKAS BOMBACH

538587

26TH AUGUST 2015

SUPERVISORS:

PROF. DR. DEBORA WEBER-WULFF

PROF. DR. BARBARA KLEINEN

HTW BERLIN

INTERNATIONAL MEDIA AND COMPUTING (MASTER)

© 2015 Lukas Bombach

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0
International License.

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 very limited and unstable. Bugs and unwanted behavior make it hard to use and can only be worked around, not fixed. By reviewing the API's history, it can be argued that its design has never been revisited only to ensure compatibility to current browsers. This thesis discusses the API's downsides and demonstrates that rich-text editing can be achieved without requiring the `contenteditable`-attribute with the library "Type", thus solving many problems of contemporary third-party rich-text editors.

Acknowledgements

I would like to extend my thanks to my two supervisors, Prof. Dr. Debora Weber-Wulff and Prof. Dr. Barbara Kleinen, 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

Contents	4
1 Introduction	7
1.1 Motivation	7
1.2 Terminology	8
1.3 Structure	8
I Theory	9
2 History of markup languages	10
2.1 Introduction	10
2.2 History of markup languages	10
2.3 HyperText Markup Language	11
3 Text editing in browser environments	13
3.1 Overview	13
3.2 Plain-text editing	13
3.3 Rich-text editing	14
3.4 HTML Editing APIs	14
3.5 Usage of HTML Editing APIs for rich-text editors	16
II Discussion	18
4 Overview	19
5 History of HTML editing APIs	20

<i>CONTENTS</i>	5
5.1 Browser support	20
5.2 Emergence of HTML editing JavaScript libraries	21
5.3 Standardization of HTML Editing APIs	22
6 Advantages and disadvantages	24
6.1 Discussion	24
6.2 Advantages of HTML Editing APIs	25
6.3 Disadvantages of HTML Editing APIs	26
6.4 Treating HTML editing API related issues	30
7 Rich-text editing without editing APIs	34
7.1 Alternatives to HTML editing APIs	34
7.2 Rich-text without HTML editing APIs in practice	38
7.3 Advantages of rich-text editing without editing APIs	39
7.4 Disadvantages of rich-text editing without editing APIs	41
8 Conclusion	44
8.1 Conclusion	44
III Concept	45
9 Approaches for enabling rich-text editing	46
9.1 Overview	46
9.2 Native inputs, images and third-party plugins	46
9.3 Enabling editing mode without using its API	46
9.4 Native text input imitation	47
9.5 Approaches for imitating native components	48
9.6 Implementation	50
10 Software design	51
10.1 Implementation as pure library	51
10.2 API	52
10.3 Distribution	55
11 Architecture	56

<i>CONTENTS</i>	6
11.1 Model–view–controller	56
11.2 Modular and object-oriented programming	57
12 Goals	59
IV Implementation	61
13 Implementation	62
13.1 Overview	62
13.2 Technology	62
13.3 Base class	64
13.4 Api	67
13.5 Input flow	67
13.6 Input reading	69
13.7 Input Pipeline	73
13.8 Pasting	76
13.9 Caret	77
13.10 Selection	79
13.11 Contents	80
13.12 Undo Manager	83
13.13 Events	85
13.14 Utility classes	87
13.15 Cache	89
13.16 Real-time collaboration with Etherpad	90
13.17 Extending	95
V Conclusion	97
14 Evaluation	98
15 Outlook	100
15.1 Stability	100
15.2 Features	100
15.3 Mobile support	100

<i>CONTENTS</i>	6
15.4 Bi-directional text support	101
15.5 Markdown editing	101
15.6 MVC & Document model	101
List of Figures	102
Listings	103
Bibliography	106
VIAppendix	110
VISource Code	ix

Chapter 1

Introduction

1.1 Motivation

Written text is the most important cultural tool to pass on knowledge from one generation to another. Designing texts, layouting, adding images and choosing text formattings is an important means to convey a message and to clarify ideas. A newspaper written as a single stream of words would not have served the purpose, that it has for hundreds of years. With the computerization of printing, PCs have been the tool to generate and design texts for decades. There are software solutions for professional and personal use on Desktop PCs. In recent years, many desktop applications have been migrated to browser-based solutions. This has many advantages. Applications can be maintained in a centralized manner and any computer using the software will be updated automatically. Browser-based applications can be accessed from anywhere in the world, without requiring to install further applications. Contents can be shared and edited collaboratively with others.

Still, rich-text editing, i.e. editing text that uses formattings and layouting, cannot be implemented easily in a browser. Browsers offer APIs for rich-text editing, but these APIs are very limited in its functionality, inconsistent across different browsers and known to contain numerous bugs.

This makes it hard for web developers to create rich-text editors. Usually, a third-party editor must be used and customized, which does not necessarily fit the specific needs of a project. The limited features of the browsers' rich-text

APIs only allow for basic editors. A fully featured word-processing application like Google's document editor cannot be implemented with these APIs. For this reason Google omitted these APIs entirely. Unfortunately, there is no library and hardly any editor that implements rich-text editing without these APIs. Google did not publish their solution to the public domain.

The purpose of this thesis is to implement rich-text editing without using the browsers' rich-text editing APIs. This allows more features, a consistent behavior and avoids the bugs of these APIs. The implementation will be distributed as GUI-less software library with a high-level API, to enable web developers to implement rich-text editors specific to their needs, which is currently not possible.

1.2 Terminology

In web development, the term *WYSIWYG* editor is commonly used to describe text-editors that allow formatting. WYSIWYG is an abbreviation for **What You See Is What You Get** and describes a text editor's capability to display formatted text as it is being edited. This stands out to plain-text editors that can neither display nor edit formattings. The term *rich-text editor* has often been used for this feature and stands in better contrast to *plain-text editor*. For this reason, the term *rich-text editor* and *rich-text editing* will be used in this thesis.

1.3 Structure

The first part of this thesis explains how rich-text editors are currently being implemented in browsers.

The second part discusses the problems with these approaches, possible alternatives as well as advantages and disadvantages of each approach.

Part three discusses techniques for an implementation of rich-text editing without rich-text editing APIs and part four discusses its implementation.

Part five gives an evaluation of this thesis.

Part I

Theory

Chapter 2

History of markup languages

2.1 Introduction

During the letterpress era, "marking up" text has been the profession of adding formalized annotations to a text, that described the structure and formattings of the document. The annotated document was given to a typesetter to follow the instructions and use a movable type system to print the document accordingly.

2.2 History of markup languages

With the computerization of typesetting, so-called "markup languages" have been invented that embedded the annotations in the text that was given to the typesetters. Coombs, Renear, and DeRose describe six types of markup languages: Punctuational, presentational, procedural, descriptive, referential and metamarkup [Coombs et al., 1987]. Punctuational markup solely refers to the use of punctuation to structure text, referential markup describes the ability of a markup language to refer to other documents and a metamarkup language can be used to describe other markup languages. Procedural markup includes commands for a computer program on how to render the text step by step. Presentational markup contains specific descriptions on the formatting of a text, describing particular parts as italicized, bold, indented etc. Descriptive markup describes the elements of a text as types. For instance, a part of a text can be marked to be a quote or a headline, but there would be no

definition on how these elements should be displayed. A renderer can parse the document and present it with specific styles. Watson describes the difference between descriptive and presentational markup as generic and specific markup. Generic markup only describes the structure of a document while specific markup explicitly describes its styling [Watson, 1992].

The invention of generic markup is credited to William Tunnicliffe who proposed his ideas at a meeting at the Canadian Government Printing Office in 1967 [Goldfarb, 1990]. Markup that was given to typesetters still needed to be translated for the particular typesetting system that was used. This led to higher costs and the need for a standard emerged [Watson, 1992]. In the late 60s, Stanley Rice, a book designer, proposed this idea of generic markup to the Graphic Communications Association (GCA), which formed the GCA GenCode committee to work on a standard for generic markup [Goldfarb, 1990]. This work has been authorized by the Organization for Standardization (ISO) and the American National Standards Institute (ANSI). In 1969 Charles Goldfarb, Edward Mosher, and Raymond Lorie invented the Generalized Markup Language (GML) [Watson, 1992], a generic markup language for IBM. Goldfarb later maintained the cooperation of ISO, ANSI and the GCA GenCode committee and in 1985 drafted the proposal for the "Standard Generalized Markup Language" (SGML), the first international standard for a generic markup language. SGML was based on the work of the GCA GenCode committee as well as the GML [Goldfarb, 1990] and published in 1986 as ISO 8879:1986 [ISO, 1986].

2.3 HyperText Markup Language

HyperText Markup Language (HTML) is an implementation of SGML [W3C, 1999, SGML and HTML]. One of the primary functions of web browsers is to display HTML formatted sources as visually formatted text. HTML uses tags to specify the contents of a document. Being an instance of SGML it mostly uses generic tags to define headlines, paragraphs or quotations inside a document, but also allows for specific tags, defining parts of the contents as italicized or bold.

Tags are strings inside the document's text that itself are delimited by

the "<" and ">" characters. *Listing 2.1* demonstrates the HTML required to render the following text:

In a hole in the ground there lived a **hobbit**

```
1 In a hole in the ground there lived a <strong>hobbit</strong>
```

Listing 2.1: Text formatted as bold with the "strong" tag

The word "hobbit" must be enclosed with a "strong" start and end tag. Start and end tags are distinguished by adding a solidus to the end tag. HTML defines 127 tags to format document contents as well as to add metadata about the document itself [Mozilla, 2015f].

By the recommendation of the World Wide Web Consortium (W3C), browsers must represent a document marked up with HTML with the Document Object Model (DOM) [W3C, 1998], a tree structure containing every tagged element and its texts as nodes. The specification of the DOM defines an API to manipulate it and change the contents of a website dynamically.

Chapter 3

Text editing in browser environments

3.1 Overview

To develop a text editor in a browser, the DOM API must be used. Development is generally restricted to the components and APIs offered by the HTML5 standard as well as experimental features that are usually implemented in a subset of browsers. The boundaries of these restrictions can be overcome. It is common practice to combine native elements and APIs in ways they have not been designed for to enable features that are not natively offered. These techniques are often referred to as "hacks" and, despite this terminology, are generally not regarded as a bad practice.

This chapter will discuss the basics of plain-text and rich-text editing in browsers as well as the APIs and techniques that browsers provide.

3.2 Plain-text editing

Text input components for browsers have been introduced with the specification of HTML 2.0 [Berners-Lee, 1995]. The components proposed include inputs for single line (written as `<input type="text" />`) and multiline texts (written as `<textarea></textarea>`). These inputs allow writing plain-text only.

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

3.3 Rich-text editing

Major browsers, i.e. any browser with a market share above 0.5%¹, do not offer native input fields that allow rich-text editing. Neither the W3C's HTML5 and HTML5.1 specifications nor the WHATWG's "HTML Living Standard"² recommend such elements. As discussed in **2.3: HyperText Markup Language**, by being able to display HTML, browsers are rich-text viewers. By the early 2000s, the first JavaScript libraries emerged, that allowed users to interactively change (parts of) a website to enable rich-text editing in the browser. The techniques used will be discussed in section 3.4 through section 3.5.

3.4 HTML Editing APIs

In July 2000, with the release of Internet Explorer 5.5, Microsoft introduced the IDL attributes³ `contentEditable` and `designMode` along with the content attribute `contenteditable` [Microsoft, 2000a, Microsoft, 2000b]. These attributes were neither part of the W3C's HTML 4.01 specification [W3C, 1999] nor the ISO/IEC 15445:2000 [ISO, 2012], the defining standards of that time. Table 3.1 lists these attributes and possible values.

```

1 <div contenteditable="true">
2   This text can be edited by the user.
3 </div>
```

Listing 3.1: An element set to editing mode

¹<http://gs.statcounter.com/#all-browser-ww-monthly-201406-201506-bar>, last checked on 07/25/2015

²The Web Hypertext Application Technology Working Group (WHATWG) is a working group that mainly developed the HTML5 standard, which later resulted in the widely acknowledged "HTML Living Standard" see **5.3: Standardization of HTML Editing APIs**

³IDL attributes can only be set to DOM objects via JavaScript, whereas content attributes can be set to tags in the HTML source code [W3C, 2012].

Order	Parameter	Description
1	cmdID	The name of the command that will be executed
2	showUI	Determines if the browser will display a dialog if needed
2	value	A parameter that can be passed to the command invoked with the cmdID

Table 3.2: execCommand parameters

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. The `designMode`-attribute can only be applied to the entire document and the `contentEditable` and `contentEditable` attributes can be applied to specific HTML elements as described on Microsoft's Developer Network (MSDN) online documentation [Microsoft, 2000b]. These elements include "divs", "paragraphs" and the document's "body" element amongst others. Other than that, there is no difference in these attributes. In editing mode

1. Users can interactively click on and type inside texts
2. An API providing commands for editing text is enabled that can be accessed via JScript and JavaScript

When an element is switched to editing mode, the browser handles setting the caret if a user clicks inside the text, accepting keyboard input and modifying text nodes entirely by itself. No further scripting is necessary.

The API enabled by the editing mode must be called globally on the `document` object, but will only execute when the user's selection or caret is contained within an element in editing mode. *Table .1* lists the full HTML editing API. To format text, the method `document.execCommand` must be used.

```
1 document.execCommand('italic', false, null);
```

Listing 3.2: Emphasizing text using the HTML editing API

Listing 3.2 demonstrates an example call of the "italic" command. Calling this at any time on the `document` object, the browser will wrap the currently selected text (if inside an element in editing mode) with `<i>` tags. The method accepts three parameters.

The first parameter is the "Command Identifier", which determines which command to execute. This can be "italic" to italicize the current selection or "createLink" to create a link with the currently selected text as label.

```
1 document.execCommand('createLink', false, 'http://example.com');
    /');
```

Listing 3.3: Creating a link using the HTML editing API

The *third* parameter will be passed on to the internal command⁴ as a parameter. In the case of a `createLink` command, the third parameter is the URL to be used for the link to create. The *second* parameter determines if executing a command should display a user interface specific to the command. Using the `createLink` command with the second parameter set to `true` while not passing a third parameter, the user will be prompted with a system dialog to enter a URL. Most commands (command identifiers) `execCommand` accepts trigger text formatting. This includes commands to format text as bold, underlined, struck-through or as a headline. A full list of possible command identifiers can be found on MSDN [Microsoft, 2015a]. Apart from executing commands, the API enabled by the editing mode includes the functions `queryCommandEnabled`, `queryCommandIndeterm`, `queryCommandState`, `queryCommandSupported` and `queryCommandValue` which allow reading attributes related to the editing mode.

3.5 Usage of HTML Editing APIs for rich-text editors

Most web-based rich-text editors use HTML editing APIs as their basis. The popular editors "CKEditor"⁵ and "TinyMCE"⁶ dynamically create an `iframe` on instantiation and set its `body` to editing mode using the `contenteditable`-attribute. This way, users can type inside the `iframe` which acts as a text input field. Both libraries wrap the `iframe` in a user interface with buttons to format the `iframe`'s contents. Using the interface, the commands of

⁴The command invoked using the command identifier

⁵<http://ckeditor.com/>, last checked on 08/22/2015

⁶<http://www.tinymce.com/>, last checked on 08/22/2015



Figure 3.1: Usage of HTML editing APIs in CKEditor and TinyMCE

`document.execCommand` will be called on the `iframe`'s `document` and the selected text will be formatted. While using an `iframe` is still in practice, many newer editors use a `div` element instead. The user interfaces vary between different editors.

Usually, rich-text editors implemented this way wrap their editing capabilities (including `document.execCommand`) in an API to enrich functionality and provide higher-level concepts. As discussed in **Part II: Discussion**, using HTML editing APIs requires a lot of workarounds (for example to fix bugs of the APIs) which some editors account for in the implementation of their library. Rich-text editing libraries can be downloaded JavaScript files and included in a web project. To display an editor on a website, it is common to select a `textarea` element on the website, that the library will replace with the rich-text editor. To integrate the editor into web forms, most libraries will mirror their contents to the selected `textarea`, so they can be submitted to a server.

For years, the market of web-based rich-text editors has been dominated by "CKEditor" and "TinyMCE". Both editors remain among the most popular choices. More recently, many new libraries have been published. Popular choices on GitHub, rated by the number of "stars", include "MediumEditor", "wysihtml" and "Summernote". As Piotrek Koszuliński points out, most editors "really doesn't[sic] work" [Koszuliński, 2013] for the reasons discussed in **6.3: Disadvantages of HTML Editing APIs**.

Part II

Discussion

Chapter 4

Overview

While HTML editing APIs are the recommended way by the W3C and the WHATWG for implementing rich-text editors on the web, their implementations across major web browsers are inconsistent, known to contain numerous bugs and have a limited and their functionality is limited and imprecise.

Understanding the origins and the history of rich-text editing on the web poses the question if the paradigms it is based on have been thoroughly reviewed and if alternative ways for an implementation, possibly using hacks, should be considered.

Chapter 5 will discuss the history and origins of HTML editing APIs. Chapter 6 will discuss its advantages and disadvantages and chapter 7 will discuss possible alternatives.

Chapter 5

History of HTML editing APIs

5.1 Browser support

As discussed in **3.4: HTML Editing APIs** HTML editing APIs have been introduced in July 2000 with the release of Internet Explorer 5.5 by Microsoft and have not been part of any standard of that time.

With the introduction of editing capabilities, Microsoft released a short documentation [Microsoft, 2000b], containing the attributes’ possible values and element restrictions along with two code examples. Although a clear purpose has not been stated, the code examples demonstrated how to implement rich-text input fields with it. Mark Pilgrim, author of the ”Dive into” book series and contributor to the the WHATWG, states that the API’s first use case has been for rich-text editing¹.

In March 2003, the Mozilla Foundation introduced an implementation of Microsoft’s `designMode`—named Midas—for their release of Mozilla 1.3. Mozilla published this as ”rich-text editing support” on the Mozilla Developer Network (MDN) [Mozilla, 2003]. In June 2008, Mozilla added support for the `contentEditable` IDL and `contenteditable` content attributes in Firefox 3.

Mozilla’s editing API closely resembles the API implemented for Internet Explorer, although, to this present day, there are still differences in the available command identifiers [Mozilla, 2015d, Microsoft, 2015b], as well as the markup generated by invoking commands [Mozilla, 2003].

¹<https://blog.whatwg.org/the-road-to-html-5-contenteditable>, last checked on 07/10/2015

In June 2006, Opera Software released Opera 9², providing full support for `contentEditable` and `designMode`³, followed by Apple in March 2008⁴ providing full support in Safari 3.1⁵. MDN lists full support in Google Chrome since version 4 [Mozilla, 2015a], released in January 2010⁶.

5.2 Emergence of HTML editing JavaScript libraries

Around 2003⁷ the first JavaScript libraries emerged that made use of Microsoft's and Mozilla's editing mode to offer rich-text editing in the browser. Typically, 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"⁸ and "TinyMCE" have been released as open source projects. These projects are still being maintained and remain among the most used rich-text editors. TinyMCE is the default editor for the content management system (CMS) Wordpress and CKEditor is listed as the most popular rich-text editor for the CMS Drupal⁹.

Since the introduction of Microsoft's HTML editing APIs, a large number of editors have been implemented. While many have been abandoned, GitHub lists about 600 JavaScript projects related to rich-text editing¹⁰. However, it should be noted, that some projects are based on other projects' editors and some projects are stubs.

²<http://www.opera.com/docs/changelogs/windows/>, last checked on 07/10/2015

³<http://www.opera.com/docs/changelogs/windows/900/>, last checked on 07/10/2015

⁴<https://www.apple.com/pr/library/2008/03/18Apple-Releases-Safari-3-1.html>, last checked on 07/10/2015

⁵<http://caniuse.com/#feat=contenteditable>, last checked on 07/10/2015

⁶http://googlechromereleases.blogspot.de/2010/01/stable-channel-update_25.html, last checked on 07/10/2015

⁷compare figures .1 and .2

⁸Now distributed as "CKEditor"

⁹https://www.drupal.org/project/project_module, last checked on 07/16/2015

¹⁰<https://github.com/search?o=desc&q=wysiwyg&s=stars&type=Repositories&utf8=%E2%9C%93>, last checked on 07/16/2015

5.3 Standardization of HTML Editing APIs

HTML editing APIs have been the *de facto* standard for implementing rich-text editors on the web, but have only been standardized in October 2014 with HTML5.

HTML5 introduces 13 new types of input fields [W3C, 2014]. It can be imagined that along with these elements, the standard could have introduced a native rich-text input element as well, but none of the elements comprises such capabilities. The WHATWG, the working group that mainly developed the HTML5 standard, discussed this issue publicly. The problems that have been faced with that idea are as follows:

1. Finding a way to tell the browser which language the rich-text input should generate. E.g. should it output BBCode¹¹, (X)HTML, Textile or something else?
2. How can browser support for a rich-text input be achieved?

Ian Hickson, editor of WHATWG and main author of the HTML5 specification, addresses these main issues in a message from November 2004¹². He states

"Realistically, I just can't see something of this scoped[sic] [the ability to specify a language for a rich-text input and possibly to specify a subset of language elements allowed] getting implemented and shipped in the default install of browsers."

and agrees with Ryan Johnson, a contributor to the standard, who states

"Anyway, I think that it might be quite a jump for manufacturers. I also see that a standard language would need to be decided upon just to describe the structure of the programming languages. Is it worth the time to come up with suggestions and examples of a programming language definition markup, or is my head in the clouds?"

¹¹A then popular markup language for bulletin boards

¹²<https://lists.w3.org/Archives/Public/public-whatwg-archive/2004Nov/0014.html>, last checked on 07/16/2015

Ian Hickson finally concludes

"Having considered all the suggestions, the only thing I could really see as being realistic would be to do something similar to (and ideally compatible with) IE's "contentEditable" and "designMode" attributes."

Mark Pilgrim lists this as a milestone of the decision to integrate Microsoft's HTML editing APIs in the standard of the WHATWG.¹³ In cooperation with the W3C, the work by the WHATWG, including the standardization of the editing APIs, have been incorporated in the HTML5 standard. The cooperation between the WHATWG and the W3C ended in Juli 2012¹⁴, which led the WHATWG to publish and maintain an own standard, the "HTML Living Standard" [WHATWG, 2015] that includes the same specifications on HTML editing APIs as HTML5.

¹³<https://blog.whatwg.org/the-road-to-html-5-contenteditable>, last checked on 07/16/2015

¹⁴<http://lists.w3.org/Archives/Public/public-whatwg-archive/2012Jul/0119.html>, last checked on 07/16/2015

Chapter 6

Advantages and disadvantages

6.1 Discussion

Understanding the history of the HTML editing APIs, the reasons for their wide browser support and their final standardization are questionable. It can be doubted if they fit their purpose specifically well. In fact, all major browsers mimicked the API as implemented in Internet Explorer 5.5, even though there was no specification for it. The reasons for this have not been publicly discussed. A reason may have been to be able to compete with other browsers. Both, Microsoft's original implementation as well as Mozilla's adoption have been released in the main years of the so-called "browser wars"¹. Mozilla adopted Microsoft's API applying practically no change to it. It can be argued that this has been part of the struggle for market shares while competing with Microsoft's Internet Explorer. At this time, it was indispensable for any browser to be compatible with as many websites as possible. A great number of websites have only been optimized for a specific browser. To gain market share, it was essential to support methods that other browsers already offered and that have been used by web developers. Being able to display websites just as good as their competitor may have been a key factor for Mozilla's decision to implement Microsoft's HTML editing APIs and not alter them in any way. Creating another standard would have been a disadvantage over the then

¹The "browser wars" was competition for market shares between Internet Explorer and Netscape Navigator during the late 1990s. Mozilla, Chrome, Safari and Opera participated as they were released in the early 2000s.

stronger Internet Explorer in getting users to choose Mozilla.

As discussed in section **5.1: Browser support**, other now popular browsers, i.e. Chrome, Safari and Opera, implemented these APIs only years later, when JavaScript libraries based on them had already been popular and widely used, which can be seen as a reason for this decision. As described in section **5.3: Standardization of HTML Editing APIs**, it has clearly been stated, that the reason for standardizing these APIs for rich-text editing has been to ensure browser support.

The API itself stems from a time when the usage of the web was different from today. JavaScript has only been standardized 3 years in advance to the publication of the HTML editing APIs. The use cases and products build with this technology are now far more complex and elaborate than of this stage of the internet. The requirements of blogging platforms or products like Google's document editor were yet unknown.

The API itself and especially its implementations across various browsers has been criticized by Google [Harris, 2010], Medium [Santos, 2014], CKSource [Koszuliński, 2013]² and others. It has led websites to exclude users from editing in certain browsers entirely³. Sections 6.2 through 6.4 discuss the advantages and disadvantages, as well as practices for treating the disadvantages of HTML editing APIs.

6.2 Advantages of HTML Editing APIs

Browser support

A fair reason for using HTML editing APIs is their wide browser support. Caniuse.com lists that 92.78% of all web users use a browser that fully supports HTML editing APIs⁴.

²The creators of CKEditor

³<https://medium.com/medium-eng/the-bug-that-blocked-the-browser-e28b64a3c0cc>, last checked on 08/19/2015

⁴<http://caniuse.com/#search=contenteditable>, last checked on 07/17/2015

High-level API

HTML editing APIs offer high-level commands for formatting text. It needs little setup to implement a basic editor, the browser takes care of generating the required markup.

HTML output

HTML editing APIs modify and generate HTML. In the context of web development, user input in this format is likely to be useful for further processing.

Possible third-party solutions for other languages

While HTML editing APIs can be used to generate HTML only, its design offers a way for third-party libraries to build on top of that and implement editors that write BBCode (for instance) and use HTML only for displaying it as rich-text. A dedicated rich-text input might not offer this flexibility.

6.3 Disadvantages of HTML Editing APIs

No specification on the generated output

The specifications on the HTML editing APIs do not state what markup should be generated by specific commands. There are vast differences in the implementations of all major browsers. Calling the `italic` command Internet Explorer, Firefox and Chrome all generate different markup.

```
1 <i>Lorem ipsum</i>
```

Listing 6.1: Markup of italic command in Internet Explorer

```
1 <span style="font-style: italic;">Lorem ipsum</span>
```

Listing 6.2: Markup of italic command in Firefox

```
1 <em>Lorem ipsum</em>
```

Listing 6.3: Markup of italic command in Chrome

This is a *major* problem for web development, because it makes processing input very difficult. For a content management system or a blogging platform, it can be very hard to handle the input of users only because different browsers are being used. Given the number of possible edge cases, it is very intricate to normalize the input.

Apart from that, Internet Explorer's output is semantically incorrect for most use cases⁵, while Firefox's output is breaking semantics entirely and is considered a bad style in terms of the separation of concerns of HTML and CSS⁶.

Furthermore, different browsers will not only generate different markup when executing commands: when a user enters a line break (by pressing enter), Firefox will insert a `
` tag, Chrome and Safari will insert a `<div>` tag and Internet Explorer will insert a `<p>` tag. Most features of the HTML editing APIs that generate markup show different implementations across different browsers.

Flawed API

The original and mostly unaltered API is limited and not very effective. MDN lists 44 commands available for their `execCommand` implementation [Mozilla, 2015c]. While other browsers do not match these commands precisely, their command lists are largely similar. 23 of these commands format the text (i.e. to italicize or make text bold) by enclosing the current selection with tags like `` or ``. The only difference between these commands is which tag will be used. At the same time there is no command to wrap the selected text in an arbitrary tag, for example to apply a custom class to a text⁷. All 23 commands could be summarized by a single command, that allows to pass custom tags or markup, that the selected text will be wrapped with. This applies to inserting elements as well. 7 commands insert different kinds of

⁵<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/i#Notes>, last checked on 07/17/2015

⁶https://en.wikipedia.org/wiki/Separation_of_concerns#HTML.2CSS.2C_JavaScript, last checked on 07/17/2015

⁷For example to apply the class "highlight" in the following manner: `Lorem ipsum`

HTML elements, this could be simplified and extended by allowing to insert any kind of (valid) markup with a single command.

Both alternatives would also give developers more control of what to insert. As previously discussed, browsers handle formatting differently. Allowing to format with specific HTML would generate consistent markup (in the scope of a website) and would allow developers generate the markup that fits their needs.

Restrictions

Google points out that implementing an editor using HTML editing APIs comes with the restriction that such an editor can only offer the least common denominator of functions supported by all browsers. They argue, if one browser does not support a specific feature or its implementation is buggy, it cannot be supported by the editor⁸. This is mostly true, although it is to be noted, that editors like CKEditor show, that some bugs can be worked around as well as some functionality be added through JavaScript. These workarounds still have limitations and not everything can be fixed. In particular there can be cases where the editing mode is not able to handle content inserted or altered by workarounds, thus limiting the features of an editor. Google names layouting the editor's contents with tab stops as one example.

Clipboard

When dealing with user input, usually some sort of filtering is required. It is possibly harmful to accept any kind of input. This must be checked on the server side since attackers can send any data, regardless of the front end a system offers. However, in a cleanly designed system, the designated front end should not accept and send "bad" data to the back end. This applies to harmful content as well as to content that is simply *unwanted*. For example, for aesthetic reasons, a comment form can be designed to allow bold and italic font formatting, but not headlines or colored text.

⁸<http://googledrive.blogspot.fr/2010/05/whats-different-about-new-google-docs.html>, last checked on 07/18/2015

Implementing a rich-text editor with HTML editing APIs, unwanted formatting can be prevented simply by not offering input controls for these formatings (assuming no malicious behavior by the user). However contents can be pasted from the clipboard that contain any kind of formatting into elements in editing mode. HTML editing APIs provide no way to define or apply filtering to the formatings of pasted contents.

Recent versions of major browsers allow observing paste events. Chrome, Safari, Firefox and Opera grant full read access to the clipboard contents from paste events. In these browsers, the event can be stopped and its contents can be processed. Internet Explorer grants access to plain-text and URL contents only. Android Browser, Chrome for Android and IOS Safari allow reading the clipboard contents on paste events as well. Other browsers and some older versions of desktop and mobile browsers do not support clipboard access or listening to paste events. Overall, 82.78% of internet users support listening to and reading from clipboard events⁹.

When dealing with the clipboard, especially older browsers show an unexpected behavior. Older WebKit-based browsers insert so-called "Apple style spans"¹⁰ on copy and paste commands. "Apple style spans" are pieces of markup that have no visible representation, but clutter up the underlying contents of an editor. When pasting formatted text from Microsoft Word, Internet Explorer inserts underlying XML, that Word uses to control its document flow, into the contents of the editor.

Bugs

HTML editing APIs are prone to numerous bugs. Especially older browser versions are problematic. Piotrek Koszuliński states:

"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 [...] are working on this for years and we still have full bugs lists [Koszuliński, 2012]"

⁹<http://caniuse.com/#feat=clipboard>, last checked on 07/18/2015

¹⁰<https://www.webkit.org/blog/1737/apple-style-span-is-gone/>, last checked on 07/18/2015

Mozilla lists 1060 active issues related to its "Editor" component¹¹. Google lists 420 active issues related to "Cr-Blink-Editing"¹². The WebKit project lists 641 active issues related to "HTML Editing"¹³. Microsoft and Opera Software do not allow public access to their bug trackers. As quoted above, some rich-text editors like CKEditor have been developed for over 10 years and still need to fix bugs related to the editing API¹⁴ [Koszuliński, 2012]. Some bugs have caused big websites to block particular browsers entirely¹⁵.

Given the argument that editing APIs provide easy to use and high-level methods to format text, in practice, the number of bugs and workarounds required, renders a "quick and easy" implementation impossible. Most importantly, browser bugs cannot be fixed by web developers. At best they can be worked around, enforcing particular software design on developers, possibly spawning more bugs and making the development dependent of the development of browsers and user adoption.

6.4 Treating HTML editing API related issues

Since the issues arising with HTML editing APIs are part of the browser's implementation, they cannot be fixed by JavaScript developers. The common approach for most rich-text editors is to use HTML editing APIs and wrap it in a library while using workarounds for its issues and bugs internally. It is to be noted, as Piotrek Koszuliński points out, that the majority of rich-text editors "really do not work" [Koszuliński, 2013]. This is usually the case when the problems discussed in **6.3: Disadvantages of HTML Editing APIs** have not been addressed and the library solely consists of a user interface wrapping an element in editing mode.

¹¹https://bugzilla.mozilla.org/buglist.cgi?bug_status=__open__&component=Editor&product=Core&query_format=advanced&order=bug_status%2Cpriority%2Cassigned_to%2Cbug_id&limit=0, last checked on 07/18/2015

¹²<https://code.google.com/p/chromium/issues/list?q=label:Cr-Blink-Editing>, last checked on 07/18/2015

¹³https://bugs.webkit.org/buglist.cgi?query_format=advanced&bug_status=UNCONFIRMED&bug_status=NEW&bug_status=ASSIGNED&bug_status=REOPENED&component=HTML%20Editing, last checked on 07/18/2015

¹⁴<http://dev.ckeditor.com/report/2>, last checked on 07/18/2015

¹⁵<https://medium.com/medium-eng/the-bug-that-blocked-the-browser-e28b64a3c0cc>, last checked on 07/18/2015

Having to account for multiple browser implementations, working around bugs can result in a big file size and a complex architecture. Most edge cases can only be learned from experience, not be foreseen or analyzed by debugging source code.

In practice, there are a few attempts to implement pure wrappers that will take care of the beforementioned issues, to support other developers with a working api. This approach is generally not well-adopted though. In general, most libraries are distributed as independent editors implementing their own solutions for addressing these issues—or are forks of other editors implemented with a different user interface.

Subsections **6.4: HTML output** through **6.4: Restrictions** will discuss some approaches to treat the beforementioned issues.

HTML output

Editors like CKEditor offer some configuration on the generated HTML output¹⁶, but in the case of CKEditor this is very limited. The underlying issue is that HTML editing APIs cannot be configured. The only way to work around this issue is to implement custom methods to apply formatting in JavaScript and not using the `execCommand` interface¹⁷. The proprietary "Redactor Text Editor" demonstrates such an implementation.

Medium.com takes a different approach and implements an extensive framework that will compare the markup of the editor with a model of the visual representation that the markup generates and corrects the DOM on each change¹⁸ to conform a defined norm.

Flawed API

HTML editing APIs are usually wrapped in the API of an editor, that offers more functionality than the original API. `execCommand` offers the `insertHTML`

¹⁶

http://docs.ckeditor.com/#!/guide/dev_output_format-section-adjusting-output-formatting-through-configuration, last checked on 08/19/2015

¹⁷HTML editing APIs can still be used for text input and other functionality

¹⁸<https://medium.com/medium-eng/why-contenteditable-is-terrible-122d8a40e480>, last checked 08/19/2015

command that allows inserting custom elements. As discussed in the previous paragraph, extending the formatting capabilities requires a JavaScript implementation.

Clipboard

For browsers, that do not offer native support to control and process the contents pasted from the clipboard, workarounds must be used. There are two approaches to this

1. Sanitize the editor's contents after a paste event.
2. Proxy a paste event to insert its contents into another element and read the contents from it.

The "Redactor Text Editor" uses the first approach. While reading contents from the a paste event is not fully supported, the event itself will be triggered by all major browsers, even most older versions¹⁹. Once the event has finished and the contents have been inserted to the editor, a "cleaned up" procedure can remove unwanted contents.

CKEditor and TinyMCE have been developed before most major browsers supported clipboard events. Both editors implement a technique to permit pasting formatted text, that has been the standard for many years. CKEditor and TinyMCE create a hidden `textarea` element and listen for common "paste" keyboard shortcuts (`ctrl` `v` and `shift` `ins`). When a user presses these keys, the hidden `textarea` will be focused and thereby be the target in which the browser will paste the clipboard's contents. After a short delay, the editors can read the `textarea`'s contents. Since `textarea` elements allow plain text only, the contents will be removed of any formatting and can then be inserted to the editor. However, this does not account for pasting from the context menu. For this CKEditor overrides the native context menu with a custom menu containing a custom "paste" menu item, that will open a modal instructing the user to paste his or her contents using the keyboard shortcuts. TinyMCE

¹⁹That have a market share BETTER CHECK THIS AGAIN

overrides the native context menu too, but does not display a paste option. Up to the current versions CKEditor²⁰ and TinyMCE²¹, this is still the case.

CodeMirror²², a web-based source code editor enhances this approach by moving the textarea to the cursor's position when the user presses his or her right mouse button. This way a native context menu can be displayed while the paste option would insert the clipboard's contents to a designated `textarea`, that can be read from.

On the downside, the paste event cannot be proxied to the `textarea` if the user uses the browser's menu bar to paste contents.

Bugs

Generally, bugs cannot be fixed. The only way to treat bugs in browsers is by avoiding them, shimming them with JavaScript methods or "cleaning up" after they have occurred.

Restrictions

The restrictions the HTML editing API imposed on the contents of the editor is an even bigger problem. Taking the example of layouting with tab stops²³, the only solution is not making the entire contents of the editor editable, but implementing a layouting engine in JavaScript and enabling the editing mode only on parts of the layout.

²⁰CKEditor 4.5.1

²¹TinyMCE 4.2.3

²²<https://codemirror.net/>, last checked on 08/22/2015

²³<http://googledrive.blogspot.fr/2010/05/whats-different-about-new-google-docs.html>, last checked 08/19/2015

Chapter 7

Rich-text editing without editing APIs

7.1 Alternatives to HTML editing APIs

Overview

HTML editing APIs are the recommended way for implementing a web-based rich-text editor. This section will discuss possible alternatives to editing rich-text.

Native input elements

Native text inputs are hard-wired to plain-text editing. No major browser offers an API for formatting. There is also no option to write HTML to an input and have it display it as rich-text. `input` fields and `textarea` elements will simply display the HTML as source code. Rich-text can only be implemented as an editable part of the website.

Image elements

In February 2015, Flipboard Inc. demonstrated an unprecedented technique to achieve fluid full-screen animations with 60 frames per second on their mobile

website¹. Instead of using the DOM to display their contents, the entire website was rendered to a `canvas` element. When a user swiped over the website the canvas element was re-rendered, essentially imitating the browser's rendering engine. `canvas` elements allow rendering rich-text too. A rich-text editor can be implemented using this technique. This however has two major downsides. On the one hand it would require implementing a text-laying engine. The `canvas` API is not capable of laying text. On the other hand, making the editor accessible to other developers would be much more complex since the text only exists in an internal representation inside the editor and would not be exposed as DOM component to other developers.

An approach related to rendering the text on a `canvas` element is to render the text inside a Scalable Vector Graphic (SVG). In contrast to `canvas` elements, SVGs contain DOM nodes that can be accessed from the outside. However this has no benefit over using HTML DOM nodes with the downside that SVG too has no native implementation for controlling the text layout.

Furthermore, while both alternatives can display rich-text, neither provides an dedicated API to manipulate rich-text, which gives neither alternative an advantage over using regular DOM structures to display rich-text.

Third-party plugins

Another way to display and edit rich-text inside a browser is through third-party plugins like Adobe Flash or Microsoft Silverlight. Flash and Silverlight lack mobile adoptions and have been subject to critique since the introduction of smartphones and HTML5. Other third-party plugins are even less well adopted. This makes Flash, Silverlight and other third-party browser-plugins a worse choice as compared to displaying and manipulating rich-text though the DOM.

Manipulation via the DOM APIs

The only way to natively display rich-text on a website is through the Document Object Model (DOM). Editors based on HTML editing APIs utilize the

¹<http://engineering.flipboard.com/2015/02/mobile-web/>, last checked on 07/24/2015

DOM to display their rich-text contents too. Only the editing (of the DOM), commonly phrased "DOM manipulation", is implemented with HTML editing APIs.

Manipulating the DOM has been possible since the first implementations of JavaScript and JScript. It has been standardized in 1998 with the W3C's "Document Object Model (Core) Level 1" specification as part of the "Document Object Model (DOM) Level 1 Specification" [W3C, 1998].

Other than for rich-text editing, the DOM and its API is the recommended² way to change a website's contents and—apart from HTML editing APIs—the only option *natively* implemented in any major browser. Popular libraries like jQuery, React or AngularJS are based on it. The API has been developed for 17 years and proven to be stable across browsers.

MDN lists 44 commands for the `execCommand` interface [Mozilla, 2015c].

- 23 commands apply text formatting.
- 6 commands insert HTML elements.
- 2 commands remove contents.
- 2 commands remove formatting.
- The other commands enable control over the clipboard, implement undo/redo commands, set settings for the editing mode and one command can select all text of the editable element.

Algorithms 1 through 4 demonstrate alternatives to commands of the `execCommand` interface related to text formatting, insertion and deletion implemented with methods of the "Document Object Model (Core) Level 1" specification.

Algorithm 1 demonstrates a simplified procedure to wrap a text selection in a tag. To implement the `bold` command of `execCommand`, this procedure can be implemented using the `strong` tag. The text selection can be read with the browser's selection API [Mozilla, 2015g]³.

²recommended by the W3C and WHATWG

³Internet Explorer prior version 9 uses a non-standard API [Microsoft, 2015c]

Algorithm 1 Simplified text formatting pseudocode

```

1: procedure FORMAT
2:    $s \leftarrow$  split text node at beginning of text
3:    $e \leftarrow$  split text node at end of text
4:    $t \leftarrow$  new tag before  $s$ 
5:   for all  $n$  in selection do                                 $\triangleright n$  is a node in the selection
6:     Move  $n$  to  $t$ 
7:   end for
8: end procedure

```

Algorithm 2 Simplified element insertion pseudocode

```

1: procedure INSERT
2:   if Selection is not collapsed then
3:      $s \leftarrow$  split text node at beginning of text
4:      $e \leftarrow$  split text node at end of text
5:     for all  $n$  in selection do                                 $\triangleright n$  is a node in the selection
6:       Remove  $n$ 
7:     end for
8:     Collapse selection
9:   end if
10:  Insert new tag at beginning of selection
11: end procedure

```

Algorithm 2 demonstrates a simplified procedure to insert a new tag and possibly overwrite the current text selection and thereby mimicking `execCommand`'s insertion commands.

Algorithm 3 Simplified text removal pseudocode

```

1: procedure REMOVE
2:   if Selection is not collapsed then
3:      $s \leftarrow$  split text node at beginning of text
4:      $e \leftarrow$  split text node at end of text
5:     for all  $n$  in selection do                                 $\triangleright n$  is a node in the selection
6:       Remove  $n$ 
7:     end for
8:     Collapse selection
9:   else
10:    Remove one character left of the beginning of the selection
11:   end if
12: end procedure

```

Algorithm 3 demonstrates a procedure to mimic the deletion commands of `execCommand`.

Algorithm 4 Simplified element unwrapping pseudocode

```

1: procedure UNWRAP( $e$ )                                ▷  $e$  is an element
2:   for all  $n$  in  $e$  do                                    ▷  $n$  is a node in  $e$ 
3:     Move  $n$  before  $e$ 
4:   end for
5: end procedure
6: Remove element

```

Algorithm 4 demonstrates a procedure to unwrap an element, mimicking the commands of `execCommand` to remove formatting.

With formatting and removing text as well as inserting and unwrapping elements, we can find equivalents for all commands of the HTML editing APIs related to manipulating rich-text using only methods specified by the "Document Object Model (Core) Level 1". This shows, that HTML editing APIs are not a necessity for rich-text editing. Chapter **13: Implementation** demonstrates ways to implement clipboard, undo/redo and selection capabilities.

7.2 Rich-text without HTML editing APIs in practice

Google completely rewrote their document editor in 2010 abandoning HTML editing APIs entirely. In a blog post⁴, they stated some of the reasons discussed in section **6.3: Disadvantages of HTML Editing APIs**. They state, using the editing mode, if a browser has a bug in a particular function, Google won't be able to fix it. In the end, they could only implement "least common denominator of features". Furthermore, abandoning HTML editing APIs enables features otherwise impossible, for example tab stops for layouting [Harris, 2010]. With the Google document editor, Google demonstrates it is possible to implement a fully featured rich-text editor using only JavaScript without HTML editing APIs.

⁴<http://googledrive.blogspot.fr/2010/05/whats-different-about-new-google-docs.html>, last checked on 07/18/2015

Google’s document editor is proprietary software and its implementation has not been documented publicly. Most rich-text editors still rely on HTML editing APIs. The editor “Firepad”⁵ is another exception. It is based on “CodeMirror”⁶ and extends it with rich-text formatting. The major disadvantage of Firepad is its origin as a source code editor. It generates “messy” (non-semantic) markup with lots of control tags. It has a sparse API that is not designed for rich-text editing and has no public methods to format the text. It is to be noted that Google’s document editor generates lots of control tags as well, but it is only used within Google’s portfolio of office apps where it may not be necessary to create *well-formatted*, semantic markup. A list of rich-text editors using and not using HTML editing APIs can be found in **Figure .1** and **Figure .2**.

7.3 Advantages of rich-text editing without editing APIs

With a pure JavaScript implementation, many of the problems that HTML editing APIs have, can be solved. The issues discussed in **6.3: Disadvantages of HTML Editing APIs** will be addressed hereinafter.

Generated output and flawed API

The generated markup, if implemented through JavaScript and DOM Level 1 methods, can be chosen with the implementation of the editor. Furthermore, the decision of the generated output can be given to the developers working with the editor. Section **6.3: Disadvantages of HTML Editing APIs** describes the inconsistent output across various browsers as well as the restrictions of the API design of `execCommand`. Both issues can be addressed by offering a method to wrap the current selection in arbitrary markup. jQuery’s `htmlString` implementation⁷ demonstrates a simple and stable way to define markup as a string and pass it as an argument to JavaScript methods. A sample call could read as follows.

⁵<http://www.firepad.io/>, last checked 07/23/2015

⁶A web-based source code editor

⁷<http://api.jquery.com/Types/#htmlString>, last checked on 07/19/2015

```

1 // Mimicking document.execCommand('italic', false, null);
2 editor.format('<em />');
3
4 // Added functionality
5 editor.format('<span class="highlight" />');

```

Listing 7.1: Example calls to format text

This will allow developers to choose which markup should be generated for italicizing text. The markup will be consistent in the scope of their project. Since the DOM manipulation is implemented in JavaScript and not by high-level browser methods, this will also ensure the same output across all systems and solve cross-browser issues. The second example function call in listing 7.1 demonstrates that custom formatting, fitting the needs of a specific project, can be achieved with the same API, giving developers a wider functionality.

7.4: Native components discusses the disadvantage, that when not using HTML editing APIs, native components like the caret or the text input must be implemented with JavaScript as they are not provided without using HTML editing APIs. On the flip side, this allows full control over these components that can be exposed via an API to other developers.

Restrictions

When implementing an editor in pure JavaScript, the limitations imposed by the HTML editing APIs, do not apply. Anything that can be implemented in a browser environment can also be implemented as part of a rich-text editor. The Google document editor demonstrates rich functionality that would not be possible with an implementation based on HTML editing APIs.

Clipboard

Without a native text input or an element switched to editing mode with HTML editing APIs, clipboard functionality is not available. Users cannot paste contents from the clipboard unless one of these elements is focused. However chapter **13: Implementation** demonstrates a way that not only allows clipboard support, but also grants full control over the pasted contents.

Bugs

By refraining from using HTML editing APIs, all of its numerous bugs will be avoided. An implementation can be aimed to minimize interaction with browser APIs, especially unstable or experimental interfaces. DOM manipulation APIs have been standardized for more than 15 years and tend to be well-proven and stable. Bugs that occur will mostly be part of the library and can be fixed and not only worked around. Bug fixes can be rolled out to users when they are fixed. This will free development from being dependent on browser development, update cycles and user adoption.

7.4 Disadvantages of rich-text editing without editing APIs

Formatting

The HTML editing APIs' formatting methods take away a crucial part of rich-text editing. Especially on the web, where a text may come from various sources, formatting must account for many edge cases. Nick Santos, author of Medium's rich-text editor states:

"Our editor should be a good citizen in [the ecosystem of rich-text editors]. That means we ought to produce HTML that's easy to read and understand. And on the flip side, we need to be aware that our editor has to deal with pasted content that can't possibly be created in our editor. [Santos, 2014]"

An editor implemented *without* HTML editing APIs does not only need to account for content (HTML) that will be pasted into the editor⁸ (in fact, content can be sanitized before it gets inserted in the editor, see **13.8: Pasting**), but also for content that will be loaded on instantiation. It cannot be assumed that the content that the editor will be loaded with (for example integrated in a CMS), is *well-formatted* markup or even valid markup. "Well-formatted" means, the markup of a text is *simple* in the sense that it expresses semantics

⁸Medium uses HTML editing APIs

with as few tags as possible (and it conforms the standards of the W3C). In HTML, the same visual representation of a text, can have many different—and valid—underlying DOM representations. Nick Santos gives the example of the following text [Santos, 2014]:

The hobbit was a very well-to-do hobbit, and his name was
Baggins.

The word "Baggins" can be written in any of the following forms:

```
1 <strong><em>Baggins</em></strong>
2 <em><strong>Baggins</strong></em>
3 <em><strong>Bagg</strong><strong>ins</strong></em>
4 <em><strong>Bagg</strong></em><strong><em>ins</em></strong>
```

Listing 7.2: Different DOM representations of an equally formatted text

A rich-text editor must be able to edit any of these representations (and more). Furthermore, the same edit operation, performed on any of these representations must provide the same *expected* behavior, i.e. generate the same visual representation and produce predictive markup. Above that, being a "good citizen" it should produce simple and semantically appropriate HTML even in cases when the given markup does not conform this rule.

Native components

As discussed in section 3.4: **HTML Editing APIs**, when an element is switched to editing mode using the HTML editing APIs, users can click inside the text and will be presented with a caret. They can move the caret with the arrow keys and enter text that will be inserted at the appropriate offset. They can use keyboard shortcuts and use the mouse's context menu to paste text. Behavior that is common for rich-text input, for instance that a new list item will be created when users press "enter" inside a list, is implemented by the browser. None of this is available when not using HTML editing APIs. All of this must be accounted for and implemented in JavaScript. Elements like the caret must be mimicked with DOM elements like the `div` element. The users'

input must be read with JavaScript and either move the caret or modify the text.

Possible performance disadvantages

Modifying the text on a website means manipulating the DOM. DOM operations can be costly in terms of performance as they can trigger a browser reflow⁹. While it should be a goal to keep browser interactions to a minimum, there is no way to avoid DOM interaction with any visual text change.

File size

While bandwidth capacities have vastly improved, there may still be situations where a JavaScript libraries' file sizes matter. This may be for mobile applications or for parts of the world with less developed connections. When not using HTML editing APIs, a lot of code must be written and transmitted just to enable basic text editing, which would not be needed otherwise.

⁹<https://developers.google.com/speed/articles/reflow>, last checked on 07/19/2015

Chapter 8

Conclusion

8.1 Conclusion

HTML editing APIs will generate different markup on most browsers, their functionality is limited and restricts web developers from extending it. As for the current state, their implementations contain plenty bugs on almost every system, which cannot be fixed by web developers. The "DOM Level 1" APIs required to perform the same tasks as HTML editing APIs have been developed and tested for more than 15 years and tend to be stable. Google's document editor demonstrates it is possible to implement a fully featured editor without using editing APIs. Doing so will avoid any restriction and limitation of these APIs and give web developers full control of all components, the generated markup and possible bugs.

Part III

Concept

Chapter 9

Approaches for enabling rich-text editing

9.1 Overview

This section will discuss the options to implement rich-text editing without relying (entirely) on HTML editing APIs and approaches to avoid their disadvantages and bugs.

9.2 Native inputs, images and third-party plugins

As discussed in **7.1: Alternatives to HTML editing APIs**, native text inputs cannot be used for rich-text editing, using image elements has no benefits and many disadvantages and third-party plugins lack user adoption. For these reasons, none of these approaches will be considered.

9.3 Enabling editing mode without using its API

One way to enable editing but avoid many bugs and browser inconsistencies, is to enable the editing mode on an element, but refrain from using `execCommand` to format the text. The latter could be implemented using the DOM core APIs. This would provide the user with all basic editing functions, i.e. a caret, text input, mouse interaction and clipboard capabilities—all of this would be taken care of by the browser.

This approach would solve the problem of buggy and inconsistent `execCommand` implementations but not the problems that arise with different browser behavior on the user's text input—for instance when entering a line break. If the markup is customly generated with JavaScript but the input would be handled by the browser's editing mode, the browser may not be able to work on the structures generated by JavaScript and break elements or simply get stuck. This was one of the reasons why Google decided to abandon editing APIs entirely¹. It could be the source to many bugs and ultimately restrict the editors capabilities.

9.4 Native text input imitation

The only other option to allow the user to change the text on a website is by manually fetching the user's input and manipulating the DOM with JavaScript and DOM Level 1 APIs. However, this does not suffice to provide the experience of a text input. The following components, common to text editing, must also be accounted for:

Caret

The caret is an essential part to text editing. Even if a user types on his or her keyboard, a caret must be seen on the screen to know where the input will be inserted. The caret also needs to be responsive to the user's interaction. In particular, the user must be able to click anywhere in the editable text and use the arrow keys to move it (possibly using modifier keys, which's behavior depends on the operating system used).

Selection

The user must be able to draw a text selection using his or her mouse and change the selection using shift and the arrow keys. Most systems allow double clicks to select words and sometimes tripple clicks to select entire paragraphs. Other systems, for example OS X, allow holding the option key to draw are rectangular text selection, independent of line breaks.

¹<http://googledrive.blogspot.fr/2010/05/whats-different-about-new-google-docs.html>, last checked on 07/21/2015

Context menu

The context menu is different in text inputs from other elements on a website. Most importantly, it offers an option to paste text, that is only available in native text inputs or elements in editing mode.

Keyboard shortcuts

Text inputs usually allow keyboard shortcuts to format the text and to perform clipboard operations. Formatting the text is possible through DOM manipulation, pasting text however only works on text inputs or elements in editing mode.

Undo / Redo

Undo and redo are common functions of text processing and it may be frustrating to users if they were missing.

Behavior

Rich-text editors (usually) share a certain behavior on user input. When writing a bulleted list, pressing the enter key usually creates another bullet point instead of inserting a new line. Pressing enter inside a heading will insert a new line. However pressing enter when the caret is located at the end of a heading commonly creates a new text paragraph after heading.

9.5 Approaches for imitating native components

These components are natively available for text inputs across all browsers. Switching an element to editing mode enables these components too. That means users can click in a text to place a caret and move it with the keyboard's arrow keys. They can copy and paste text. The browser offers a native context menu that allows pasting on input elements as well as on element in editing mode. All major browsers implement a behavior for the users' input that is common for rich-text editing.

When not using editing APIs, all of this must be implemented with JavaScript. This requires a lot of trickery and many components must be imitated to make

it *seem* there is an input field, where there is none. The users must be convinced they are using a native input and must not notice they are not.



Figure 9.1: Rendering of highlighted source code in Ace and CodeMirror

Web-based code editors like "Ace"² and "CodeMirror" demonstrate that this is possible. They display syntax-highlighted source code editable by the user. The user seemingly writes inside the highlighted text and is also presented with a caret as well as the above mentioned components. In reality, the content that the user sees is a "regular" part of the DOM—non-editable text, colored and formatted using HTML and CSS. When the user enters text, the input will be read with JavaScript. Based on the input Ace and CodeMirror generate HTML and add it to the editors contents, to show a properly syntax-highlighted representation (see figure 9.1). A `div` element that is styled to look like a caret is shown and moved with the user's keyboard and mouse input. The user's text input will be inserted at the according text offset. Amongst others, Ace and CodeMirror use DOM elements like `divs` to display a text selection and a *hidden textarea* to fetch keyboard inputs, to recreate the behavior and capabilities of a native text input.

Using tricks and *faking* elements or behavior is common in web front end development. This applies to JavaScript as well as to CSS. For instance, long before CSS3 has been developed, techniques have been discussed on how to implement rounded corners without actual browser support. Only years later, this has become a standard. This not only enables features long before the creators of browsers implement them, this *feedback* by the community of web developers also influences future standards. Incorporating feedback is a core philosophy of the WHATWG, the original creators of HTML5.

²<http://ace.c9.io/>, last checked on 08/22/2015

9.6 Implementation

The library implemented for this thesis uses similar techniques as Ace and CodeMirror to create a rich-text editor. The contents of the editor are represented by a `div` element that contains the formatted text using HTML. The caret as a `div` element styled to mimic a native caret. The text selection is also displayed by using `div` elements that are styled accordingly. The keyboard and mouse input will be read with JavaScript and the contents of the editor will be changed accordingly using DOM Level 1 methods. The user's input will also be read to move the mimicked caret on the website with JavaScript. The specifics on the implementation of these components and how they interact will be discussed in **13: Implementation**.

Chapter 10

Software design

10.1 Implementation as pure library

Most rich-text editors are implemented and distributed as user interface components. That means instead of only providing a library that offers methods to format the selected text and leaving the implementation of the user interface to the respective developer, most libraries are distributed as input fields with a default editor interface that is, at best, customizable.

This can be unfitting for many situations. The user interface of an editor highly depends on the software it will be integrated in. Within the software the interface may even vary depending on its specific purpose. For instance, a content management system may require an editor with a menubar offering many controls while a comment form on a blog requires only very little controls. Medium.com uses an interface that only shows controls when the user selects text and has no menubar at all. Assuming there are many implementations of editors that are functional, it can be argued, that choosing between editors is often really a choice of the desired user interface.

Customizing a user interface can be just as complex as writing an interface from scratch. The latter affords to add HTML elements and call JavaScript methods while both require styling. While adding HTML elements to a website is not a complex task, in a worst case scenario, it can be more complicated to customize an interface to specific needs than writing an interface from scratch and being able to define just the elements as they are required.

As for the current state of the internet, web developers cannot easily implement a rich-text for themselves, they have to make a choice between pre-made solutions and customize them. Apart from the perspective of the user interface, integrating a fully featured editor into a software project can be invasive to the structure of the project.

For these reasons the library of this thesis will be implemented and distributed as a pure software library, offering developers an API to create a rich-text editor, rather than a fully implemented rich-text editor as a user interface component.

10.2 API

The library should be capable of any method implemented by HTML editing APIs. However the API design can differ to improve the way it will be worked with. In particular the API aims at providing a quick and simple way to create editable areas and connecting a user interfaces to it.

API Design

The API of this library must be *well-designed*. That means it must be simple, effective and fit the developers' needs. The methods it offers should be simple in the sense that they conceal possibly complex tasks with understandable high-level concepts. They should be effective and fit the developers' needs in the sense that the API should be designed so that any requirement of the developers should be matched with as little effort as possible. The API should create a workflow for developers that allows them to do what they intend to do and is as easy to use and as plausible as possible. jQuery is an example of incorporating an API that comes close to these goals.

The library's API will have two basic use cases. On the one hand, web developers must be enabled to implement rich-text editors with it. On the other hand, the library should offer interfaces for enabling web developers to extend the library and add features.

Extension For extension, web developers should have precise access to as many components and functions of the library, providing as much freedom and

options as possible. This will include low-level access to components while control and explicitness is more important than simplicity.

All components of the library will be implemented as classes. To provide as much capabilities as possible to other developers, all classes of the library will be exposed in a designated namespace. The classes should conform the best practices of object-oriented programming to support developers in extending the library. The class design should not only consider the specific needs of the core library but also potential use cases for other developers.

For example, with a designated class to show and move a caret, multiple carets can be instantiated for an extension that allows real-time collaboration with multiple users. All available classes will be discussed in chapter **13: Implementation**.

Editor implementation For web developers implementing an editor, the API should be designed to offer methods for the most common tasks related to rich-text editing to allow fast and easy integration in a website. This should be high-level methods as compared to methods required for extending the library. Simplicity is more important than precise control over low-level behavior. For implementing a rich-text editor the exposed methods should cover

1. Formatting and removing formats
2. Insertion
3. Deletion
4. Controlling the caret
5. Controlling the text selection
6. Controlling the clipboard
7. Controlling settings
8. Undo / redo commands

jQuery demonstrates an effective and simple approach to API design, conforming the principles as discussed above. In jQuery all methods remain in a flat

hierarchy within the root of a jQuery collection. Any method that is not a getter allows chaining and most methods are overloaded to allow passing various kinds of parameters, to determine what the function should do. Following these and the above-mentioned principles, the components listed above can be expressed in 11 functions:

```
1 editor.caret([options]);
2 editor.selection([options]);
3 editor.insert([options]);
4 editor.format([options]);
5 editor.remove([options]);
6 editor.settings([options]);
7 editor.copy();
8 editor.cut();
9 editor.paste();
10 editor.undo();
11 editor.redo();
```

Listing 10.1: API for implementing a rich-text editor

The functions in lines 1 through 6 can take various overloaded parameters to determine the specific action. The selection command, for instance, can be called with two numbers to draw a selection from one character offset to another. To draw a selection from characters 10 to 20 `editor.selection(10, 20)` can be called. The function can also be called without passing any parameters to read the selection. `editor.selection()` will return the currently selected contents. A full API description can be found in tables .2 through .8.

Handling use cases

We can call programmers extending the library "developers of the library" and programmers using the library to implement editors "users of the library". To account for both use cases and maintain a clear software architecture as well as a separation of concerns, all classes that provide functionality to the library must remain in a designated namespace which the library has access to. Developers of the library have access to the namespace and can utilize any of its classes to extend its functions.



Figure 10.1: Diagram of the Type library and its internally used classes (excerpt)

The classes within the namespace will be used by a globally accessible class called "Type" which is the entry point for the users of the library to implement rich-text editors. The Type class provides an API with all of the above-mentioned methods and uses the classes inside the namespace for their implementation. It must be instantiated and be passed an element on the website (for example a `div` element) which it will then use as its "editor contents". The users of the library can build an interface for this editable element and use the instance's API to edit its rich-text contents.

10.3 Distribution

The library will be distributed as a single JavaScript file. Extensions by third-party developers will each be distributed as independent and separate (JavaScript) files. By exposing Type and its classes as discussed in section **10.2: API** they can be accessed from other files. This provides a modular "plug and play" system for distributing and loading extensions. To improve loading times, web developers can concatenate Type and its extensions to a single file in a web project.

Chapter 11

Architecture

11.1 Model–view–controller

Model–view–controller (MVC) is a common approach for implementing user interfaces and it can be applied to user interface components too. While this approach can provide clear responsibilities, the problem is that most components, like the caret or the selection, serve a clear atomic purpose and would need to be broken apart into model, view and controller parts themselves, making the architecture fuzzy and complex instead of simplifying it.

Following the MVC architecture, the contents of the editor (the text) can be represented in a model (holding the text data and allowing methods to be performed on) and be rendered with a view (displaying the text in the browser). In contrast to the beforementioned components, this would be a very clean model for implementing the editor’s contents. It is even imaginable to implement multiple renderers in the view layer, turning the editor from rich-text into a Markdown editor, for instance.

Unfortunately, this approach would make the contents of the editor only editable through the API of the ”Type” library. If any other script on a website would change its contents, the library’s renderer would overwrite the changes with the next rendering the data of the internal model. As discussed in **10.2: API**, the library shall leave as much freedom as possible to the developers. This would create a bottleneck and restrict other developers. For this reason, the MVC architecture will not be used.

11.2 Modular and object-oriented programming

jQuery and CKEditor demonstrate a software architecture in which a base object, which is exposed to other developers as the library, provides an environment to extend its functionality, but does not offer many methods itself¹. The actual functionality of both libraries is implemented through extensions while the libraries are usually bundled with a set of "core extensions" that provide basic features. CKEditor makes use of modular programming techniques by implementing a major part of its editor as plugins that communicate via strictly defined interfaces. jQuery established a paradigm calling any extension a "plugin" but instead of using strictly defined interfaces, developers are encouraged to add arbitrary methods to jQuery's base object, which can then be directly accessed. Extending a base object has many advantages:

1. It provides a namespace for the library
2. It provides a structure for extensions to access each other
3. It approaches modular programming and strong decoupling

Strict modular programming could create a system in which other developers can exchange any component easily to improve performance or enrich functionality. The disadvantage this approach would be that the need for well-defined interfaces can diminish flexibility. Formalizing interfaces would create complex structures and could make it harder for other developers to contribute to the library instead of inviting them. jQuery uses another approach and encourages arbitrary extensions. jQuery's approach demonstrates that this flexibility, in practice, can withstand possible conflicts. In turn, the low barrier for extending jQuery has spawned a rich collection of extensions and a big community of developers. While jQuery technically allows to be extended with complex libraries, it is designed to be extended with simple methods. It is difficult to establish complex interactions between extensions.

¹CKEditor provides a framework for implementing components for it, but does not offer any rich-text functionality in its core. jQuery provides low-level utility methods for JavaScript.

Constructor Pattern & modularized structure

To close the gap between CKEditor’s modular programming approach and jQuery’s simple extension paradigms, object-oriented programming (OOP) can be used. JavaScript does not offer classes and classical inheritance, however the same functionality can be achieved using the constructor pattern and prototypal inheritance (see **13.14: OOP**). Functions following the constructor pattern are often called classes or pseudo-classes. Hereinafter the term classes will be used.

As discussed in **10.2: Handling use cases**, the base class will be globally accessible with the name "Type". It will provide the namespace for classes that extend the library and implement its functionality. A set of core extensions will provide all components needed for a rich-text editor. The "Type" base class can be instantiated and will be the entry point for *users* of the library (see **10.2: Handling use cases**) to implement a rich-text editor. Like CKEditor and jQuery, will implement as little functionality as possible itself. The implementation of the base class as well as the interaction of its extensions will be discussed in detail in chapter **13: Implementation**. Implementing the library’s extensions as classes has many benefits:

1. As compared to CKEditor and modular programming, strictly defined interfaces are not a necessity. This can improve flexibility and lower the barrier for other developers to contribute.
2. As compared to jQuery, classes can have complex interfaces, which allows rich functionality and possibilities in interaction.
3. Classes are a proven concept for encapsulating functionality and data, protecting access and structuring code as well as making it readable.
4. Through JavaScript’s prototypical inheritance, the class can be instantiated as often as desired, but will only be allocated once in the browser’s memory. Thereby the performance will be improved. Instance variables still allow to reuse a class in different contexts with different inherent data.

Chapter 12

Goals

The library of this thesis shall enable web developers to implement rich-text editors themselves. The following features shall be implemented:

Writing and deleting text Users of an editor shall be able to type inside the text using hardware as well as virtual keyboards. Behavior to support the users' input as discussed in **9.4: Native text input imitation** will be an optional "nice-to-have" requirement.

Text formatting Users shall be able to format the text. It must be distinguished between inline and block formattings. Block formattings break the text flow and always create new lines before and after the formatted text. Block formattings include headings, paragraphs and quotations amongst others. Rich-text editors often show a different behavior for applying block formattings as compared to inline formattings. Block formattings will change any text block (i.e. block-formatted parts of a text) affected by the formatting command into the format that is being applied, even if only a part of a text block has been affected. Inline formattings will strictly apply the formatting to the part of the text that has been selected to be formatted. The text shall be formattable with any HTML tag and HTML attributes.

Caret Users shall be able to control a caret with a mouse and the arrow keys on a keyboard.

Selection Users shall be able to create a text selection and copy its contents.

Undo and redo Users shall be able to use keyboard shortcuts to undo and redo changes they made.

Clipboard control Developers shall define which contents can be pasted from the clipboard and apply rules to remove unwanted formattings.

Real-time collaboration The library shall provide basic real-time collaboration between editors over the network.

Media Editing images and other non-text content is not part of this thesis.

Browser support The library shall be stable in at least one major browser.

Part IV

Implementation

Chapter 13

Implementation

13.1 Overview

As discussed in **11.2: Constructor Pattern & modularized structure**, Type’s implementation relies on a base class that provides a high-level API for implementing rich-text editors. The library’s functionality is implemented through various other classes encapsulated in a designated namespace. Section **13.2: Technology** will discuss the tools used to develop and build the library. Section **13.3: Base class** will discuss the base class and the namespace it creates. Sections **13.5: Input flow** and following discuss the functionality, architecture and the classes involved in Type by explaining how the user’s input will be read, processed and written to the website.

13.2 Technology

Overview

There are no pre-made solutions or conventions suggesting how to design, structure and concatenate components for a JavaScript library. The most popular JavaScript libraries on GitHub¹ each implement custom and different solutions. Angular.js implements a custom module-system and uses the tool ”Grunt” to concatenate multiple files into one. D3.js uses a Makefile and var-

¹<https://github.com/search?l=JavaScript&q=stars%3A%3E1&s=stars&type=Repositories>

ious Node.js modules for building and concatenation. jQuery uses Grunt for concatenation and runs custom scripts implemented using Node.js to manipulate and clean up the resulting source code file. To support development, the library is split up into multiple files. One file each contains one class. The tools to concatenate, build and check the sources will be discussed in the following sections.

Gulp

Gulp is a Node.js-based task runner that is widely used as a build system for web applications. For this library, Gulp tasks are used to lint the source code, check code style rules and to concatenate and minify the library into a single file, that can be distributed to other developers.

Building

CommonJS specifies a system to define a so-called module in a JavaScript file that can be loaded by other modules by referencing the file name. The module that will be returned when loaded by other modules can be an arbitrary object or a primitive data type. For this library, each file is written as a CommonJS module containing and returning a single class. All files will be concatenated using RequireJS² in a designated Gulp task to a single file. The resulting file contains code structures generated by RequireJS that will allow the modules that have been defined across multiple files to access each other within a single file. This will increase the library's file size. AMDclean³ is used to remove as much of this supporting code as possible while maintaining the functionality. To further decrease the file size UglifyJS2⁴ is used to shorten variable names, remove whitespace and compress the file using "gzip".

Linting

JSLint is used to lint the source code before any concatenation happens. It will also check if the code conforms the widely acclaimed JavaScript code

²<http://requirejs.org/>, last checked on 8/18/2015

³<http://gregfranko.com/amdclean/>, last checked on 8/18/2015

⁴<https://github.com/mishoo/UglifyJS2>, last checked on 8/18/2015

conventions introduced by Douglas Crockford in his book "JavaScript: The Good Parts" [Crockford, 2008]. The code mostly conforms these conventions, differing in the way the constructor pattern is implemented to favor better readability. Classes using the constructor pattern are implemented using the conventions of the Ace library and use the prefix convention⁵ for private members.

Code formatting

JSCS is used to check the code to conform specific formatting conventions—for instance the number of spaces used for indentation or a consistent use of CamelCase. Along with JSLint checking for Douglas Crockford's conventions this ensures a consistent coding style across all files and classes.

13.3 Base class

Overview

The **Type** class is the base class (see **11.2: Modular and object-oriented programming**) is the starting point for users and developers of the library. It provides 4 purposes:

1. It can be instantiated to offer a high-level API to manipulate text and perform other rich-text related functions.
2. It provides a namespace for the library's internal classes.
3. A Type instance provides mutual access for the instances of the internally used classes.
4. It exposes its **prototype** as a public shorthand attribute.

Instantiation and usage

To develop an editor with the library, the **Type** base class must be instantiated. As discussed in **10.2: Handling use cases**, all of the library's functionality is

⁵https://developer.mozilla.org/en-US/Add-ons/SDK/Guides/Contributor_s_Guide/Private_Properties#Using_Prefixes

provided by this class. It exposes its features as high-level instance methods. The `Type` class is exposed to the `window` namespace and is thereby globally accessible. On instantiation it must be passed an `HTMLElement`, which's contents will act as the editor's rich-text content and all rich-text operations will be performed on it. Just like with HTML editing APIs, developers can build user interfaces around the editable element. An optional second parameter can be passed to the class to define settings for the editor's instance.

```
1 var element = document.getElementById("myElement");
2 var editor = new Type(element, { paste: "text" });
```

Listing 13.1: Type instantiation

The `editor` instance variable now offers methods to format, insert and remove text, manipulate the caret and the selection, dynamically change settings and control undo/redo capabilities as well as trigger clipboard commands. The complete API is listed in tables .2 through .8. For example, to format the characters 10 to 20 as bold and move the caret behind the formatted text, the following methods can be executed:

```
1 editor.selection(10, 20);
2 editor.format("<strong />");
3 editor.caret(20);
```

Listing 13.2: Example commands to format text

This is just an example to demonstrate the API. It should be noted that the API allows to specify a text range in the format command as well as chaining and the above code can be simplified to a single line.

```
1 editor.format("<strong />", 10, 20).caret(20);
```

Listing 13.3: Example chaining

With the second optional parameter, settings can be passed to the `Type` class on instantiation to determine the editor's behavior. As for the implementation of this thesis two settings are available: To determine the behavior on paste events and to turn off default keyboard shortcuts. Passing options to

the editor can be useful for extensions which can access any option passed be accessing the **Type** base class.

Namespace and references

The **Type** class creates a namespace for other classes used in the library. In JavaScript, functions are first-class objects, namespaces are objects and any object is a namespace. This way, the **Type Function** object (the class itself) can act as a valid namespace. CKEditor takes the same approach and attaches each module to the **CKEDITOR** base class. Any of the classes listed in figures .3 through .6 are attached to the **Type** class this way. As an example, listing 13.4 shows the declaration of the **Type** base class as well as the declarations of the classes **Caret**, **Range** and **Environment**:

```
1 // Declaration of the Type base class
2 function Type() {};
3
4 // Classes defined within the namespace created by Type
5 Type.Caret = function () {};
6 Type.Range = function () {};
7 Type.Environment = function () {};
```

Listing 13.4: Declaration of **Caret**, **Range** and **Environment** classes

The namespace provides a structure for the classes of the library, prevents the pollution of the global namespace and possible name conflicts. In terms of structure, this does not only encapsulate classes related to the library but also allows nesting. This way sub-namespaces can be created, which is especially important for other developers extending the library. The **Type** base class already creates a sub-namespace **Type.Events** for events.

On instantiation, the **Type** class, in turn, will instantiate classes of its namespace that implement its rich-text editing functionality. The **Type** instance will pass a reference to itself to all classes and offers getters for every class instance it created, to provide mutual access for each class of the library.

Exposal of `Type`'s prototype

While `Type`'s functionality is implemented through classes within its namespace, this does not expose its functionality to its instance-API. With the constructor pattern, all methods in the `prototype` of the `Type` class will be available as instance methods. In a simple approach, the library's API can be implemented directly in the implementation of the `Type` class. This contradicts the modular approach of the library. jQuery established an effective principle for extending its API. It exposes the library's `prototype` with a shorthand attribute as `jQuery.fn`. This way, other modules can extend the `prototype` easily and add methods to the library's API. `Type` follows the same principle and exposes its `prototype` as `Type.fn`. The `prototype` could also be accessed without exposing it with a shorthand attribute, but this is intended to clarify its purpose similar to jQuery and encourage developers to extend it.

13.4 Api

The methods of `Type` instances can be added by any of the classes of the library. However, to achieve a clear separation of concerns, maintainability and to conform the modularized structure, `Type`'s instance API is implemented through a designated module that adds all methods to the API.

The module for this, called "CoreApi", is the only exception in `Type`'s implementation—it extends `Type.fn` directly with all methods listed in tables .2 through .8, *without* using a class.

13.5 Input flow

To enable reading the user's input and writing it to the editor's contents, the classes `Input`, `Contents`, `Writer`, `Formatter`, `Caret` and `Selection` will be instantiated by the `Type` base class.

The `Input` class will listen to keyboard input and mouse input. It is responsible for setting the caret and the selection using the `Caret` and `Selection` classes and uses them to determine which part of the text should be changed or formatted when the user enters text, uses keyboard shortcuts or uses his or her mouse or touch device. It passes formalized edit operations to the `Contents`



Figure 13.1: Components instantiated by the `Type` base class

class which will emit events for an `UndoManager` that enables undo and redo operations. The `Contents` class uses the `Writer` and `Formatter` instances to manipulate the visible text on the website. These classes perform the actual DOM operations on the contents of the element passed to `Type` on instantiation (see **13.3: Instantiation and usage**).

Usually, text input fields contain one caret and display one text selection at a time. For this reason the `Type` base class instantiates the `Caret` and `Selection` classes for shared usage within an editor's instance. Of course, this behavior can be extended, for example by instantiating multiple `Carets` for real-time text collaboration.

13.6 Input reading

There are various input methods with which users can interact with native inputs. This includes using hardware devices as well as virtual (on screen) devices:

- Hardware keyboard input
- Virtual keyboard input
- Mouse input
- Touch input
- Game controller input (on game consoles)
- Remote control input (on smart TVs)

When mimicking a native input, in a best-case scenario, all these input methods should be accounted for. Fetching input includes two scenarios: The user clicks, touches or focuses the input in any way and does so at any position inside the input. If the user points (touches, clicks, etc.) in the middle of the text, the caret should move to that position. In environments without hardware keyboards, the library must ensure that a virtual keyboard shows up. Once the input is focused, text input must be fetched and written to the contents. There are various options to fetch user input, which will be discussed in the following sections.

Events

One way to fetch user input is by listening to events.

Keyboard Text input can be read through `KeyboardEvents`. Keyboard events will be triggered for virtual keyboards and for hardware keyboards. When the user presses a key, the event can be stopped and the according characters can be inserted at the offset of the caret. As a downside, listeners for keyboard events cannot be bound to an element that is not a native text input, that means keyboard events must be listened to on the `document` level.

This does not only have (minor) performance downsides but also requires more logic to decide whether a keyboard input should be processed and ultimately stopped or ignored and allowed to bubble to other event listeners of a website. In particular, there can be edge cases, where even though a keyboard event should write contents to the editor, the event itself is supposed to trigger other methods that are not part of the editor. Keyboard events are supported by all major browsers across all devices.

Mouse and touch To support clicking or touching inside the editor's contents `MouseEvent`s and `TouchEvent`s can be used. Mouse events are supported on all major desktop browsers and all mobile browsers support touch events. Both event types support reading the coordinates indicating where the click or touch has been performed.

Remote controls Although some smart TVs offer keyboards, mice, pointers similar to Nintendo's Wii remote, input via smartphone apps and many other input devices, button-based remote controls are offered with almost any smart TV and remain an edge case for interacting with a text editor. In such an environment, users commonly switch between elements by selecting focusable elements with a directional pad. Only using events would not account for this since there would be no focusable element representing the editor. Recent browsers on Samsung's and LG's smart TVs are based on WebKit⁶ while Sony's TVs use Opera. Before 2012 Samsung's browser was based on Gecko. All of these browsers and browser engines support keyboard events triggered by virtual keyboards to fetch their input.

Clipboard Another problem with relying entirely on events is the lack of native clipboard capabilities. Unless a native text input (including elements with enabled editing mode) is focused, shortcut keys for pasting will not trigger a paste event and the mouse's context menu will not offer an option for pasting.

⁶<http://www.samsungdforum.com/Devtools/Sdkdownload>, last checked on 07/22/2015

Hidden native input fields

As discussed in **9.5: Approaches for imitating native components**, the source code editors Ace and CodeMirror use a hidden (native) input field to fetch the users' keyboard input. While it appears to the users they are entering text in a syntax-highlighted representation of the source code, in reality users enter their text in a *hidden* `textarea` element. The input will be read from the `textarea`, processed and displayed with syntax-highlighting using HTML. This solves many problems that occur with relying solely on events:

- The hidden `textarea` can be focused with the tab key.
- The hidden `textarea` can be focused with remote controls.
- Virtual (on screen) keyboards will show up when the `textarea` is focused.
- Keyboard shortcuts for clipboard events work.
- It can display a native context menu that allows pasting.

Implementation

The `textarea` is created when the editor gets instantiated. Since browsers scroll the `textarea` into view when it receives the focus, it is positioned in the visual representation of the editor, scrolling the editor into view⁷. This perfectly mimics the browser's native behavior. To maintain the illusion that the user actually writes inside the visual representation of the editor the `textarea` is hidden.

Focus

Whenever the user clicks inside the editors visible contents, the mimicked caret will be moved (see **13.9: Caret**) to the according text position. To enable text input, the hidden `textarea` will be focused on click or touch events. The `textarea` is natively focusable using the tab key or a remote control on a smart TV. It will also trigger focus and blur events. This way, it is possible to display

⁷This does not mean the editor will be scrolled into view on instantiation, but when the user focuses it, for example with the tab key.

the caret when the `textarea` receives the focus and read its input as well as hiding the caret on blur and thereby perfectly mimic the native behavior for input events.

Virtual (on screen) keyboard support

The `textarea` will be focused when the user clicks or touches inside the editor as well as with the tab key and remote controls. Focusing a text input triggers the display of native virtual keyboards.

Pasting

When a the `textarea` is focused, pasting via keyboard shortcuts is natively available. To enable pasting with the context menu, CodeMirror implements a technique where the `textarea` will be moved to the pointer's position on a `mousedown` event. Following the order of `MouseEvent`s, this will be completed before the context menu will be triggered. This way it will be triggered on the `textarea` and contain a paste option. The paste event will insert the contents from the clipboard to the `textarea` from which the contents can be read.

Reading input

`textarea` elements support `input` events which can be used to read the text entered by the user. The input can be processed as discussed in **13.5: Input flow** and be removed from the `textarea`. In practice, this means that once a single character has been entered in the `textarea` it will be read from the `textarea`, inserted into the editor's contents and the `textarea` will be cleared again. Input reading requires further processing before it can be passed on to trigger a change in the editor's contents. The specifics on processing the input will be discussed in section **13.7: Input Pipeline**.

Editing mode

Using a `textarea` element allows plain-text input only. This is not a problem for regular keyboard input but rich-text contents pasted from the clipboard will be inserted as plain text and all formatings will be removed. To come

around this issue, a `div` element in editing mode can be used instead of a `textarea` element.

As discussed in **Part II: Discussion** HTML editing APIs are very problematic and a key factor of this thesis is implementing a rich-text editor without using them. However using an element in editing mode only for input reading is not affected by these issues. Whenever a single character will be entered it will be read and immediately removed from the editable element. Formatting commands will not be used at all. Problematic text input behavior, for instance different markup that will be generated by the entering a line break, will not occur since the the editable area will only be used for reading input, the text that will be inserted in the editor will be generated by the library (see **13.7: Input Pipeline**). The only difference between a `textarea` and the editable element lies in the different contents it accepts for pasting. **6.3: Clipboard** discusses the problem that text pasted from the clipboard cannot be processed with native APIs across all browser. In this case, the clipboard contents will be pasted to a designated field from which it can be read, isolated from the editors contents, which solves this problem (see section **13.8: Pasting**).

13.7 Input Pipeline

Overview



Figure 13.2: Input pipeline with sample filters

Before the text read from the hidden input field will be passed on to the `Contents` class, it will be passed through a series of `Filters`, called the "input pipeline". The input pipeline has 3 basic responsibilities.

- Stop and dispatch input that that should trigger functions of the library
- Implement rich-text editing behavior

- Filter and transform text

The input pipeline is part of the **Input** class. The pipeline itself is an array of input filters that can be added, removed and ordered with a designated API.

Input filters have a public API that specifies for which input they should be called. For instance, a filter can specify to be called when the user presses the **ctrl****s** key combination. A filter can specify a handler for the input. For pressing **ctrl****s**, a filter can specify to call a "save" function. The input is passed to the filters as an event (see **13.13: InputEvent**) that can be stopped from bubbling. This way it can be prevented that pressing **ctrl****s** will also insert an "s" character to the editor. The order in which filters will be called is important. Some filters process the input and must cancel the event not only to prevent a character to be inserted, but also to prevent other filters from taking action.

The basic filters implemented for this library will be listed hereinafter, grouped by the responsibility as listed above.

Triggering functions

Caret Commonly, when a user presses one of the arrow keys inside a text, the caret will be moved and of course, in a browser environment, this is not different. To mimic this behavior arrow key input will be intercepted by the **Caret** input filter class, which will move the caret that has been instantiated by the **Type** base class (see **13.5: Input flow**). It will also account for modifier keys and the operating system used and move the caret accordingly.

Command The **Command** input filter class checks for and intercepts keyboard shortcuts commonly used for text formatting.

- **ctrl****b** Formats the currently selected text bold.
- **ctrl****i** Formats the currently selected text italic.
- **ctrl****u** Formats the currently selected text underlined.

To format the text, by default, the tags **strong**, **i** and **u** will be used. Since in some cases this might not be desired, these default keyboard shortcuts must

be opted-in by setting an option on instantiation (see **13.3: Instantiation and usage**).

The input event implements an abstraction to either check for the `ctrl` key or the `cmd` key depending on the operating system of the user (see **13.13: InputEvent**) so that the above-mentioned shortcuts can (only) be triggered with the `cmd` key instead of the `ctrl` key on the OS X systems.

Rich-text behavior

As discussed in **9.4: Native text input imitation**, most rich-text editors support the user with a common behavior reacting to the user's input. This behavior can be abstracted in a simple way using the input pipeline. The following paragraphs describe how the behavior discussed in section 9.4 can be implemented using filters. Being a "nice-to-have"-feature, other filters was given more priority and these filters have not been implemented.

Headlines When a user presses `enter` while the caret is located at the end of a headline, a new text paragraph can be created behind the headline and the caret can be placed inside it.

Lists Pressing `enter` inside a list item can create a new list item behind the current item and the caret can be placed inside it.

Filtering and transformation

Line Breaks To display a line break in the contents of the editor a `br` or `p` tag must be inserted. When the user presses the `enter` key it does not suffice to insert a carriage return and/or line feed. This input will be intercepted and instead a `br` tag will be inserted to the editor's contents. As discussed in **13.7: Overview**, it is important that this filter will be invoked after the **Headlines** and **Lists** filters, so they can apply their behavior and prevent this filter's behavior.

Remove To delete text from the editor the **Remove** filter checks for `backspace` and `del` key inputs. Depending on whether there is a text selection or not, it either deletes the selection's contents or one character left/right of the caret.

Spaces Browsers display adjacent spaces as a single space. This is an unusual behavior for text editing. The **Spaces** input filter checks if adjacent spaces are being entered and inserts non-breaking spaces.

Events

As an alternative approach input filters could be implemented as input event handlers. With this, the same functionality could be achieved without an input pipeline. A designated pipeline however provides a clearer mental model for processing the input as it allows a separation of concerns. An input filter has a specific purpose and context as compared to an arbitrary input event handler. It can also be made sure that filters will be called in the right order and that any input filter will be run before triggering an input event. This way input event handlers only receive actual text input for the editors contents while keyboard shortcuts and other keypresses have been filtered out.

Extendability

The input pipeline is intended to be extended. It serves as an entry point for other developers to process input. For this, the **Input** class provides an API to add, remove and reorder input filters.

13.8 Pasting

As discussed in **13.6: Hidden native input fields**, all text input will be read from a designated input field. It is useful to distinguish between regular keyboard input from input pasted from the clipboard since the clipboard can contain rich text contents. Developers implementing an editor with Type should be able to determine which formattings should be allowed in the editor, i.e. pasted from the clipboard. This requires two steps.

1. Determine if an input has been made through typing or pasted from the clipboard.
2. Process the clipboard contents and make them accessible to developers.

Paste detection As discussed in **6.3: Clipboard**, modern browsers trigger paste events, which can be listened to. Not all browsers allow reading contents from a paste event, but this is not necessary. The paste event will insert its contents into the designated input element from which its contents can be read, after the event has completed. Some older browsers, specifically Opera versions older than 12.1 do not trigger paste events at all. For legacy support, the browser can be tested for an available clipboard API and in case it is missing, the text input can be checked for its text length. With the system discussed in **13.6: Hidden native input fields**, an input will always have the length of a single character. If the input is longer than that, this either means more than one character has been inserted or a single formatted character has been inserted⁸. These cases can only happen when contents have been pasted from the clipboard. However, if a single unformatted character has been pasted, it cannot be distinguished from a regular text input. It is to be noted that the use case this feature is designed for, is to sanitize pasted input, which is not necessary for a single plain text character input, although it must be acknowledged that there can be use cases requiring to register any paste event for other reasons.

Processing To process the pasted contents and possibly prevent inserting the contents to the editor an `InputEvent` will be generated and passed through the input pipeline. Any filter can be implemented to treat or ignore paste events. Users of the library can set an option on instantiation to determine how to treat pasted contents. These options include to allow plain text only, to allow any formatted text or specifying rules to allow specific formattings only. A full API description can be found at ABC. These options are implemented in the `Paste` filter, that will either let any contents pass through (allow any formatting) or filter out specific or all HTML tags.

13.9 Caret

The `Caret` class provides all functionality to place and move a caret in a text. It provides methods to be moved left, right, up and down in a text as well as

⁸The input field will contain markup of more than one character

to be placed at a specific position in a text. The visual representation of the caret is a `div` element, styled to imitate a text caret. Using a CSS3 animation, it imitates the "blinking" common for native text carets.

The elements for carets as well as for the text selection will not be written to the editor's contents. The editor's contents should not contain any markup other than for the text itself. Instead this and all other elements will be stored in a designated `div` element at the end of the website's `body` and be positioned using CSS.

The challenge with this class is that it must be able to be moved within text and in any kind of formatting, represented by any combination of DOM nodes. To be moved across letters and text lines, the caret must take into account that:

1. Letters have different widths and heights
2. Different fonts have different letter dimensions
3. Different formattings like a headline, italicized text or text with a specifically set font size, result in different letter dimensions

CodeMirror solves these problems by measuring each letter with the use of text ranges. Browsers offer a **Range** interface, a construct that has a start- and end-offset in a text. A range has methods to read its x- and y-coordinates on the website. These methods can be used to span a range over a single character, read its offsets and place the caret next to it using CSS and giving it the same height as the character.

To move the caret left or right, the according characters left and right of its current offset will be measured using this method. To move it up and down across text lines, the caret must check the offsets of every character, starting from the character of the current offset, until it reaches the character above or below it that is closest to its horizontal position. As discussed in section **13.15: Cache**, a cache to store positions cannot be applied. The complexity of this is method $O(n)$, however in practice, the number of characters this will affect is limited by readability and usability of the text editor. Mobile devices, that generally have less performance than desktop machines, have smaller screens

displaying less characters per line. While this is not necessarily the case, it can be expected by good software design.

13.10 Selection

Using a designated input element for input reading comes with the cost of having to emulate the text selection. When the input field is focused, any selection on the web site, including that of the editor, will be removed. When text is selected, the input field does not necessarily have to be focused. To read inputs, it can be focused on a "keydown" event, which will only remove the text selection when the user enters text. This is not problematic since selections will be removed on native inputs when a user enters text too. However, if the user right-clicks in the editor, the input element will be focused to enable pasting from the context menu (see **13.6: Pasting**). This will remove the text selection on any right click.

The W3C specifies an API to add multiple ranges to a selection, which should appear as multiple selections to the user. This way the element for input reading could be focused while other parts of the website, i.e. the editor's contents, could display a selection at the same time. However, while the API is available across all major browsers, it is dysfunctional and documented to not be working.

CodeMirror, ACE and Google's document editor each implement text selections by displaying `div` elements that mimic the look of a native selection. Type uses the same technique to show a text selection while the input element is focused. This mimicked selection replaces native selections entirely and will be created dynamically when the user clicks in the text and drags his or her mouse across the text.

The downside of this technique is that copy commands will not work anymore due to the fact that there is no actual text selection that can be copied, even though it appears to the users there is one. To treat this issue, CodeMirror adds the contents of the imitated selection to the hidden input field and selects these contents with a native selection. This allows the user to use keyboard shortcuts at any time and to copy text with the context menu. When the user types, the selected contents in the input field will be overwritten by the

browser, so this does not affect input reading. Type uses the same technique.

13.11 Contents

The **Contents** class provides an API to add, remove and format text. This functionality is implemented through the **Writer** and **Formatter** classes. Its central responsibility is to proxy commands to these classes and to create "actions" to pass them on to the **UndoManager**. An action describes the formatting, change or deletion of contents in a formalized way that can be undone by the **UndoManager** (see **13.12: Undo Manager**).

Writer

The **Writer** class implements functionality to add and remove contents to and from the editor. Along with the **Formatter** class, this is the lowest layer of the editor that will perform DOM operations to modify the contents in the browser.

Formatter

The **Formatter** is one of the key classes of the Type library. As discussed in **7.4: Formatting** it must generate *well-formatted* markup while being able to work on any *ill-formatted* markup it will be given. There is a virtually infinite number of edge-cases for markup that formatting commands can be applied to. Assume we have the following string.

```
1 <p>Lorem ipsum <em>dolor<u> sit amet</u></em> consec</p>
```

Listing 13.5: Markup with highlighted target for formatting

Listing 13.5 represents markup for the formatted string "Lorem ipsum dolor sit amet". The highlighted part (yellow) represents the part of the text that should be formatted using a formatting command.

Figure 13.3 shows the DOM representation of the markup of *Listing 13.5*. We can split up the text node "Lorem ipsum" into two text nodes "Lorem" and " ipsum" to create a distinct node in which the formatting (yellow highlight)

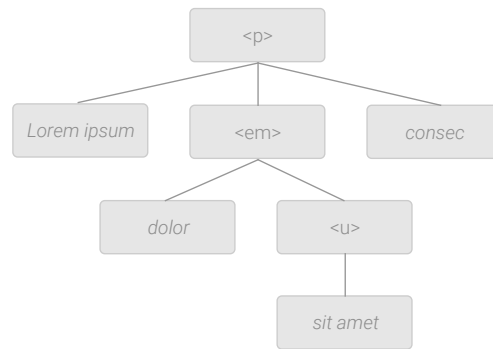


Figure 13.3: DOM representation of figure 13.5

starts and do the same with the ending text node "sit amet". This gives us a distinct nodes for the start and the end of the selection. When we traverse each node from the start to the end, every traversed node falls on either of the following cases:

1. It is the start node
2. It is the end node
3. It is a node between start and end node (but is not and does not contain the start or end node)
4. It contains end node

To generate *well-formatted* markup as discussed in **7.4: Formatting**, the following 2 rules must be followed:

1. The markup must conform the validation rules of HTML as specified by the W3C.
2. The markup must use as little DOM nodes as possible.

Algorithm 5 demonstrates a recursive algorithm, that will start iterating from the given start node over all its subsequent siblings until there are no siblings anymore *or* it found the end node *or* it found a node containing the end node.

Algorithm 5 Recursive algorithm to apply text formatting

```

1: procedure FORMAT( $s, e$ )  $\triangleright$   $s$  and  $e$  are the distinct start and end nodes
2:    $c \leftarrow s$ 
3:   Let  $a$  is an empty array
4:   while  $c \neq \text{null}$  and  $c \neq e$  and  $c$  does not contain  $e$  do
5:      $a.\text{push}(c)$ 
6:      $c \leftarrow c.\text{nextSibling}$ 
7:   end while
8:   if  $c = e$  then
9:      $a.\text{push}(c)$ 
10:  end if
11:  if  $c$  contains  $e$  then
12:    FORMAT( $c.\text{firstChild}, e$ )
13:  end if
14:  if  $c = \text{null}$  then
15:     $n \leftarrow$  next node in document flow
16:    FORMAT( $n, e$ )
17:  end if
18:  Wrap all nodes from  $a$  with DOM node to apply formatting
19:  Connect siblings to wrapping node if they have the same tag
20: end procedure

```

- If the end node was found, the algorithm will wrap all all nodes it found with a node that applies the desired formatting.
- If there are no more siblings, it means it has reached the last sibling inside the node containing the start node. By the validation rules of the W3C, nodes are not allowed to intersect. The algorithm will wrap all nodes it found so far with a node that applies the desired formatting and recursively applies itself with the start node being the next node in the document flow and the end node remaining the same end node.
- If an element has been found that contains the end node, the algorithm will wrap all nodes it found so far with a node that applies the desired formatting (to avoid intersecting nodes). It will then apply the formatting algorithm recursively to the first child of the containing node. If this node in return is another container to the end node, the recursion will repeat until a sibling of the end node or the end node itself has been found.

Each rule performs the minimally necessary steps to format contents while conforming the HTML validation rules of not intersecting DOM nodes. By adding only the minimum of nodes, this will ensure simple (and valid) markup.

To format the nodes that have been collected by the algorithm, they will be wrapped by a DOM node that applies the desired formatting. The wrapping function will also remove any nodes between the start and end node that have the same tag as the node used for the formatting to clean up the markup. If the start or the end node has been contained by a node of the same tag as the node the text should be formatted with, the containing nodes will be used as start or end node. As a last step to improve the markup, the nodes left and right of the formatting node will be unified with the formatting node, if they have the same tag. These steps will simplify potentially *ill-formatted* markup that the formatting command affects. All DOM manipulations will be performed at the end of the algorithm, when all nodes have been read to improve performance.

13.12 Undo Manager

The `UndoManager` implements the undo and redo functionality of the editor. It can receive "actions", which are instances of classes inheriting from the `Type` action base class. Each instance has the methods `execute` and `undo` to apply and revoke its particular functionality. There are 3 types of actions.

Insert

The `Insert` action will insert text on execution and delete the text when its `undo` method will be called. It utilizes the `Writer` class for these operations. It must be instantiated with the according text so it can be executed and undone at any time.

Remove

The `Remove` action will remove text on execution and store it to insert it again when it is being undone. Just like the `Insert` action it uses the `Writer` class for this.

Format

The **Format** action applies formattings using the **Formatter** class. It will store references to the nodes the **Formatter** created to remove them when the action will be undone.

Stack

The **UndoManager** stores each action to an array (the undo stack) so that each action can be undone and redone in order. In some cases, consecutive actions, like multiple insertions, should be undone in a single undo command invoked by the user. For this, each action must implement the methods **mergable** and **merge**. When an action get added to the **UndoManager** the **mergable** of the highest element in the undo stack will return if it accepts the new action to be merged with it. The **Insert** action will accept other instances of the **Insert** action class so that consecutive character input can be undone in chunks and not only in characters. Each action will implement merging with other instances in its **merge** method to which a new action will be passed if **mergable** returned **true**. Otherwise the new action will be added to the undo stack. The **UndoManager** will only merge actions that have been passed within a time frame of 500 milliseconds, to generate a distinct undo history.

Action sources

The Etherpad extension (see **13.16: Real-time collaboration with Etherpad**) allows multiple users to modify the same text. The undo methods must only undo changes from the user that initiated the action. Furthermore, undoing and redoing an action must account for changes that affect the action, for instance, when the offset at which a text has been inserted has shifted, because another user has inserted text before that offset. To manage this, each action contains a source identifier that will be generated by the **UndoManager**. With this identifier the **UndoManager** can choose to undo actions of a particular user and account for changes of actions from other users.

13.13 Events

Overview

It is possible to trigger custom (native) events using the `CustomEvent` interface on modern browsers [Mozilla, 2015b]. Internet Explorer 9 and below allow this through similar interfaces. These interfaces could be used for triggering events for components of `Type`. However, this would trigger events that are only relevant within the library in the global namespace. To avoid this, `Type` implements its own event system that populates events only within the library. Events from within the editor that can be useful to the website or web application should still be triggered as native events in the browser’s global namespace.

Event Api

The `EventApi` class provides an API to add and remove event listeners as well as to trigger events. It provides instance and static methods.

```

1  // Static methods
2  EventApi.on(eventName, eventHandler);
3  EventApi.off(eventName, eventHandler);
4  EventApi.trigger(eventName, eventObject);
5
6  // Instance methods
7  EventApi.prototype.on(eventName, eventHandler);
8  EventApi.prototype.off(eventName, eventHandler);
9  EventApi.prototype.trigger(eventName, eventObject);

```

Listing 13.6: EventApi methods

Using the OOP class (see **13.14: OOP**), these methods will be inherited by the `Type` class. This way, using the `trigger` method, events can be triggered within the scope of a `Type` instance and be observed using the `on` method. Event handlers can be removed using `off` method.

The static methods will also be inherited by the `Type` class. This is necessary to trigger events that are *Type* specific but not *instance* specific. Most importantly a "ready" event will be triggered on every instantiation of the `Type`

class. This way plugins and other third-party scripts can run initialization routines.

Plugins and third-party libraries can trigger arbitrary events and pass along arbitrary data. As a paradigm and in terms of a *good programming style*, event objects should be passed. Event objects are inherited from the `TypeEvent` or conform its API.






TypeEvent

```
1 // Gets or sets data
2 TypeEvent.data([options]);
3
4 // Stops the event from bubbling
5 TypeEvent.cancel();
```

Listing 13.7: TypeEvent API

The `TypeEvent` is a generic, general-purpose event. It can store arbitrary data and offers an API to be stopped from bubbling.

InputEvent

The Input `InputEvent` will be triggered by the `Input` class after a keyboard input has passed the input pipeline. It inherits all methods from the `TypeEvent` to conform the event system and contains information on the key and the modifier keys pressed. The key is represented with its key code and a key name. The key name will be mapped from the key code and is implemented with a list of readable names including "backspace", "enter", "space" and others. The list not complete but can be extended during further development. The modifier keys include "shift", "alt", "ctrl" and "meta". "meta" is the browser's name for the  key on OS X systems. OS X uses the  key as modifier the same way Windows and Linux use the  key. To support developing an editor for both platforms the "cmd" modifier will be set when a user holds the  key on OS X *or* the  key on other platforms.

PasteEvent

The `PasteEvent` will be triggered when the user pastes contents from the clipboard *before* it will be inserted to the editor's contents. It contains the clipboard's contents and can be cancelled so that other developers are free to manipulate and stop a paste event.

13.14 Utility classes

OOP

The `OOP` class extends the constructor pattern with basic classical inheritance. It provides the method `inherits` that will duplicate and copy the `prototype` from one `Function` object to another. It also copies attributes and methods defined on the `Function` object itself to implement inheritance of static definitions. It adds the attribute `_super` to the inheriting `Function` object referencing its parent class to enable child classes to access their respective superclasses.

Range

The `Range` class is an abstraction for the native `Range` interface. The native implementation is prone to bugs on many browsers. Instead of fixing the API by shimming its methods, the `Range` class implements all methods related to ranges while trying to interact as little as possible with the native API. On top of the methods of the native `Range` interface, this class implements additional methods required for `Type`.

Dom Walker

Working with text implies having to traverse the DOM, i.e. the nodes inside the text often. The `DomWalker` utility class solves this problem. The DOM API offers methods to access a node's siblings, children and parents, but it must always account for cases when any of these are `null` (there is no parent, sibling or child) or when they overflow the bounds of the editor's contents. But more importantly, for text editing, it is usually necessary to access the

next (or previous) node in the document's content flow which can either be the parent, sibling, child or a node that can only be accessed by traversing multiple nodes. Also, it is often the case that it is not only necessary to fetch the next node, but to apply a filter to only fetch a specific node, for instance a text node or only a text node that has contents visible to the user⁹. Browsers offer a native API for this, called **TreeWalker**, but it is said to be slow¹⁰, only partially supported by Internet Explorer 9 [Mozilla, 2015e] and has been criticized for its verbose API¹¹.

The **DomWalker** can be instantiated by providing a starting node and the type of nodes it should traverse. The latter argument can either be a string identifying a pre-made filter of the **Type** library or a custom filter function. Pre-made filters include:

- "text" - A text node with visible contents
- "textNode" - Any text node, visible or invisible
- "visible" - Any visible DOM node

A **DomWalker** instance offers the high-level methods **next**, **previous**, **first** and **last** (amongst others) for traversal.

Text Walker

The **TextWalker** class acts as a container for all functions related to measuring text offsets. It provides utility methods to determine the character offset from one text node to another or, vice versa, which text node can be found at a text offset, starting from another given node. Both methods are required by various classes and are thus, centralized.

Dom Utilities

The **DomUtilities** class encapsulates common methods for all DOM operations other than traversal. It has no inherent purpose but many other classes

⁹Any text node consisting of whitespace only will not be displayed by any major browser

¹⁰<http://jsperf.com/qa-vs-node-iterator>

¹¹By John Resig, author of jQuery, <http://ejohn.org/blog/unimpressed-by-nodeiterator/>, last checked 08/19/2015

perform the same DOM operations, which hence reside in a common library to avoid code duplication.

Utilities

The **Utilities** class is a general-purpose class that contains methods to extend JavaScript's features. It contains methods to work with data structures and to detect object types.

Environment

The **Environment** class checks and provides informations on the current browser environment and its features. This class is especially important to mimic native behavior for user interaction. For instance, as discussed before, either the control key or the command key should be used to implement keyboard shortcuts depending on the operating system. To check for specific feature support, it is favorable to use duck typing within each class.

Settings

The settings class stores settings required for Type's modules, for instance the `id` of the DOM-container which all helper DOM-nodes from other classes will be appended to.

Development

The **Development** class is intended to contain utility methods to support the development of the library. As for the development of this thesis, it was sufficient to implement logging methods.

13.15 Cache

For traversing the text, for example when the caret moves, the text will need to be measured. All measurements can be stored to a cache to only perform the same measurement operations once. When the user edits the contents of the editor, these texts will change and the cache must be updated. A cache must also account for external changes. The DOM3 Events specification [W3C,

2015] offers `MutationObservers` to check for DOM changes. This feature is not supported by Internet Explorer version 10 or less¹². Internet Explorer 9 and 10 offer an implementation for `MutationEvents`¹³. The W3C states that "The `MutationEvent` interface [...] has not yet been completely and interoperably implemented across user agents. In addition, there have been critiques that the interface, as designed, introduces a performance and implementation challenge." [W3C, 2015, Legacy `MutationEvent` events]. For this reason, the editor does not use any caching. Implementing an editor that is stateless in regards of its contents can also improve stability.

13.16 Real-time collaboration with Etherpad

Overview

To achieve real-time collaboration with multiple Type editors, Etherpad¹⁴ can be used. Etherpad is a web-based collaborative real-time text editor with rich-text capabilities. It provides a server, written in JavaScript using Node.js as well as a web-based rich-text editor. Both components are distributed in one package and are meant to be used together.

To achieve real-time rich-text collaboration, multiple web-based clients communicate with a server via WebSockets using `socket.io`. Each client owns a local version of the document that it needs to sync with the server. The server uses an operational transformation algorithm to merge each change to the document accounting for all changes and then urges each client to update their local contents according to the final document.

Changesets

For this, Etherpad uses the concept of so-called "changesets". Each client sends its local changes, debounced to an interval of 500 milliseconds, as a serialized string—the changeset—to the server. The changeset includes all text

¹²<http://caniuse.com/#search=mutation>, last checked on 07/21/2015

¹³<http://help.dottoro.com/ljfvvdm.php#additionalEvents>, last checked on 07/21/2015

¹⁴Etherpad has been completely rewritten under the name Etherpad Lite. However, its official website no longer links to its former source code. For simplicity, the name Etherpad will be used, referring to its rewrite as Etherpad Lite

insertions, removals and formattings of the last time frame. Along with the changeset, it sends a document revision number that the changeset is based on to the server. The document revision number increases with every changeset that has been accepted and applied to the document on the server side. The document on the server side is saved as a stack of changesets, which ultimately form the current document. For performance reasons, snapshots can be taken that save the document as formatted text.

Based on the revision number that the client provides with the changeset, the server can apply it to the version of the document the client was working on. The server will apply the resulting changes to all newer revisions of the document (if present) and send a changeset and the latest revision number back to the client. The changeset sent to the client includes all operations it needs to perform to update its local version to the newest version on the server.

As a last step, the client must apply the changeset it got from the server to its local document to display the most recent version to the user and update its local revision number to what it got from the server.

Merging

In a collaborative environment, it can happen that two (or more) clients send different changesets to the server that are based on the same document revision. It is the responsibility of the server to merge both changes so that it preserves either intent. As explained in the "Etherpad and EasySync Technical Manual"[citation needed], to solve this, for a document X with the conflicting changesets A and B , the server computes the new changesets A' and B' such that

$$XAB' = XBA' = Xm(A, B)$$

where $Xm(A, B)$ is the merge of A and B applied to the document X . The changesets A' and B' will be sent to the respective clients, which will apply it to their local documents to sync with the document on the server. To compute a changeset A'

- Insertions in B become retained characters in A'

- Insertions in A stay insertions in A'
- Retain whatever characters are retained in *both* A and B

For B' this applies vice versa.

Etherpad Client implementation

Clients interact with the server via WebSockets using socket.io. To sync their own changes with other clients, a client does 4 things.

- Request a the full document from the server
- Send a changeset to the server
- Receive acknowledgement from the server for a submitted changeset
- Receive a changeset from the server submitted by another client

When Etherpad's client connects to the server it receives an initial snapshot of the entire document as a string. To submit changes, the client uses a three-step architecture. The client stores any local changes that have not been sent to the server yet in a changeset Y . Any changeset sent to the server must be acknowledged by the server as it has applied the changeset to its document. Any changeset that has been sent and not been acknowledged yet will be stored in a as the changeset X . The document as it is acknowledged by the server is stored in a changeset A . The document visible to the users can be expressed by the representation of $Y \cdot X \cdot A$, i.e. applying each changeset to the next.

Whenever a user applies a local change, the changeset Y will be updated. Every 500 milliseconds, but not before a changeset submitted to the server as been acknowledged, the changeset Y will be sent to the server and Y will be assigned to X . Y will be set to a changeset that contains no changes. When the client hears the acknowledgement for X from the server, X will be applied the changeset A and X will be set to contain no changes.

This architecture supports receiving changesets from other clients as they must be applied to a client's local changes (committed and uncommitted) as well as the document version as acknowledged by the server. When a client receives another client's changeset B it will perform 4 steps.

1. Compute a new changeset by merging $Y(X \cdot B)$ and apply it to the document visible to the user.
2. Apply B to A .
3. Compute a new changeset by merging B and X and assign it to X
4. Compute a new changeset by merging $(X \cdot B) \cdot Y$ and assign it to Y

The operations needed to merge the changesets on the client, are the same operations for merging changesets on the server.

Type Client implementation

Etherpad’s technology can be used to enable real-time collaboration for Type. While Etherpad offers a web-based client, its implementation has three flaws:

1. It cannot be integrated easily in other web applications.
2. It does not generate semantic markup. It is cluttered with control sequences.
3. It is hard to extend.

Etherpad does not provide a documentation on its client–server protocol, but it can be reverse engineered. It is possible for third-party libraries to communicate with an Etherpad server alongside Etherpad’s ”native” clients, as long as a third-party library (like Type) conforms the protocol.

Etherpad’s collaboration functionality comes with a cost in file size for Type and may only be used in specific use cases. This is why this feature is implemented as an optional extension (compare **13.17: Extending**) in a separate file. To enable collaboration the designated JavaScript file needs to be added to the website.

```
1 <script src="type.js"></script>
2 <script src="type.etherpad.js"></script>
```

Listing 13.8: Enabling real-time collaboration to Type

`type.etherpad.js` adds the classes it requires to the `Type` namespace and adds a static constructor to the `Type` library:

```
1 var element = document.getElementById("myElement");
2 var editor = Type.fromEtherpad(element, "http://example.com/
  editor/myEditorId");
```

Listing 13.9: Static constructor to generate a collaborative `Type` instance

The constructor used in line 2 of *Listing 13.9* will connect to an Etherpad server and append the contents from the server to the element given as first parameter.

`Type`'s implementation for collaboration is simpler than Etherpad's native implementation. The `EtherpadClient` class sends and receives changes to and from an Etherpad server. The `EtherpadChangeset` class translates changesets from the server to edit commands for the local editor and vice versa. A changeset can contain multiple commands. For every command there is class representing it. Text insertions are represented by the `Insertion` class, deletions are represented by the `Removal` class and formattings are represented by the `Formatting` class. Every class implements a function to apply its changes to the editor as well as to return a fragment of a changeset representing its change.

When a changeset will be received it will be translated to these classes and they will be applied to the editor. When the local user changes the contents, the according class will be instantiated and added to a `EtherpadChangeset` class. Every 500ms this changeset will be serialized and sent to the Etherpad server using the fragments as returned by the command instances.

When a submitted change had to be merged with another change on the server, `Type`'s client will receive a change like any other change from other clients and it will be applied to the local editor.

This architecture provides an unobtrusive way to integrate real-time collaboration in the `Type` library. It does not depend on a specific implementation of an editor. Developers are free to implement any editor specific to their needs with integrated real-time collaboration.

13.17 Extending

Overview

Type’s modular structure is designed for extension. Type’s `prototype` has been exposed as `Type.fn` and all its classes in the `Type` namespace. This provides other developers with all of Type’s functionality in a structured and accessible manner. Type is designed to lower the barrier for and encourage developers to extend Type by giving freedom and possibilities in how to implement an extension and trying to avoid compulsorily use of interfaces or configurations.

jQuery demonstrates a similarly liberal approach for writing extensions and experience shows that name conflicts are minimal and “good” extensions are naturally favored over “bad” extensions by the community of web developers.

API extension

```
1 Type.fn.myMethod = function () {};
```

Listing 13.10: Example Type instance API extension

As discussed in **13.3: Exposal of Type’s prototype**, to add a method to Type’s public API, its base class’ prototype can be extended with a function using the `Type.fn` shorthand attribute. Static constructors can be added by extending the `Type Function` object.

```
1 Type.myConstructor = function () {  
2   return new Type();  
3 };
```

Listing 13.11: Example custom static constructor

Namespace extension

As discussed in **13.3: Namespace and references**, to implement extensions for Type, the `Type` namespace can be used to add custom classes or sub-namespaces.

```
1 Type.MyClass = function () {  
2   var caret = new Type.Caret();  
3 };
```

Listing 13.12: Example Type namespace extension and usage of a built-in class.

All other classes that Type uses are exposed in this namespace and can be used by extensions.

Plugin API

A plugin may need to be initialized when an editor will be instantiated. To support this, Type will trigger an event on instantiation and pass the Type instance to the event handler

```
1 Type.on('ready', function(typeInstance) {});
```

Listing 13.13: Example event handler for a Type instantiation

To store and read data specific to an instance, Type offers the **data** method, that will return an **Object** for arbitrary access.

```
1 Type.fn.myMethod = function () {  
2   this.data("myPlugin").foo = 'bar';  
3   var bar = this.data("myPlugin").foo;  
4 };
```

Listing 13.14: Example calls to format text

To give each plugin an own namespace, an arbitrary identifier must be passed as a **String** to the **data** method, which will provide a unique **Object** for different string identifiers. This can possibly cause name conflicts if two plugins choose to use the same string. Developers are advised to always use their own extension name as identifier. Experience with jQuery's plugin system as well as jQuery's **data** method shows that while this cannot prevent name conflicts, it is rarely a problem.

Part V

Conclusion

Chapter 14

Evaluation

The library of this thesis demonstrates a way to implement a web-based rich-text editor without using HTML editing APIs. Developers can instantiate editable areas and manipulate its contents with an API. They can enable their users to:

1. Type and delete text.
2. Apply formattings.
3. Place and move a caret.
4. Create a selection.
5. Copy and paste text.
6. Use undo and redo commands.

Unfortunately, not all features are perfectly stable and not all of the goals could be met. On the client side, the selection shows glitches and formatting commands are not fully implemented. They do not distinguish between inline and block formattings and tags, to format the content with, cannot contain attributes. This means that text cannot be styled with CSS, which takes away many formatting options, and that links cannot be added. The contents pasted from the clipboard cannot be sanitized yet.

An editor that is connected to an Etherpad server will add strings sent from other clients as well as remove text that has been removed by others.

It can receive text formattings, although this throws errors in some cases. Text that has been added and removed locally will be added and removed on other clients, although the algorithm breaks on new lines (i.e. text cannot be changed after a new line) and formattings will not be sent.

The best experience can be achieved on Google Chrome, other browsers require further testing.

The library lays a foundation to work with rich text and manage all components that are necessary for it. It provides a rich set of classes to improve and extend its functionality. The Etherpad extension, enabling real-time collaboration that can be used with any editor implemented with the library, is an example of that.

Chapter 15

Outlook

The library implements basic features of rich-text editing. In order to distribute the library for productive use some features need be completed and added.

15.1 Stability

Currently, basic rich-text editing is possible, but is unstable in some edge cases. The current features need to be tested and fixed for all major desktop browsers.

15.2 Features

The library only supports a basic set of rich-text editing features. The goals that have not been met—most importantly distinction between block and inline formattings and formattings with attributes—must be implemented. Above that, features known from other rich-text editors, especially adding media must be added.

15.3 Mobile support

The APIs the library uses for editing and formatting the text are available on mobile browsers, but interaction with touch devices has not been implemented.

It should not be difficult to add support for touch events and thereby add mobile support for the library.

15.4 Bi-directional text support

The library does not support bi-directional text yet. The Unicode Consortium published an algorithm [UC, 2015] on how to derive the visual order from a unicode string containing bi-directional text, which can be used to move the caret accordingly.

15.5 Markdown editing

The library can be used to implement a syntax highlighter for Markdown using input filters. For example, an input filter can insert a headline when a user types a hash mark at the beginning of a line or italicize text between two asterisks. This functionality can be distributed as an optional extension.

15.6 MVC & Document model

Towards the end of this thesis, Marijn Haverbeke, author of CodeMirror, started a crowd-funding campaign for supporting him to build a rich-text editor that does not rely on HTML editing APIs (called ProseMirror¹) just like the library of this thesis. As discussed in **11.1: Model-view-controller**, Type does not rely on an architecture that abstracts the contents of the editor from the DOM. ProseMirror, by contrast, takes this approach. This restricts working with the editor to the capabilities of the internal model of the document and might make extending the editor complicated. On the other hand, it allows for a very stable rendering of the contents and switching between HTML or Markdown output. It is good to see both approaches being realized. By contrasting both ideas in a practical manner it might be easier to decide which approach is better for which purposes.

¹<http://prosemirror.net/>, last checked on 08/24/2015

List of Figures

3.1	Usage of HTML editing APIs in CKEditor and TinyMCE	17
9.1	Rendering of highlighted source code in Ace and CodeMirror	49
10.1	Diagram of the Type library and its internally used classes (excerpt)	55
13.1	Components instantiated by the Type base class	68
13.2	Input pipeline with sample filters	73
13.3	DOM representation of figure 13.5	81
.1	Editors using HTML editing APIs (selection)	iv
.2	Editors not using HTML editing APIs (selection)	iv
.3	Classes of the library 1/3	v
.4	Classes of the library 2/3	vi
.5	Classes of the library 3/3	vii
.6	Classes of the Etherpad Extension	viii

Listings

2.1	Text formatted as bold with the "strong" tag	12
3.1	An element set to editing mode	14
3.2	Emphasizing text using the HTML editing API	15
3.3	Creating a link using the HTML editing API	16
6.1	Markup of italic command in Internet Explorer	26
6.2	Markup of italic command in Firefox	26
6.3	Markup of italic command in Chrome	26
7.1	Example calls to format text	40
7.2	Different DOM representations of an equally formatted text . .	42
10.1	API for implementing a rich-text editor	54
13.1	Type instantiation	65
13.2	Example commands to format text	65
13.3	Example chaining	65
13.4	Declaration of Caret, Range and Environment classes	66
13.5	Markup with highlighted target for formatting	80
13.6	EventApi methods	85
13.7	TypeEvent API	86
13.8	Enabling real-time collaboration to Type	93
13.9	Static constructor to generate a collaborative Type instance . .	94
13.10	Example Type instance API extension	95
13.11	Example custom static constructor	95
13.12	Example Type namespace extension and usage of a built-in class.	95
13.13	Example event handler for a Type instantiation	96
13.14	Example calls to format text	96
1	core.js	xi

2	development.js	xiii
3	settings.js	xiv
4	oop.js	xiv
5	event_api.js	xiv
6	plugin_api.js	xvi
7	environment.js	xvii
8	utilities.js	xvii
9	dom_utilities.js	xviii
10	dom_walker.js	xxi
11	text_walker.js	xxvii
12	range.js	xxviii
13	writer.js	xxxvi
14	formatter.js	xxxviii
15	caret.js	xli
16	selection_overlay.js	l
17	selection.js	liv
18	input.js	lviii
19	events/type.js	lxii
20	events/input.js	lxiii
21	input_filters/caret.js	lxiv
22	input_filters/undo.js	lxiv
23	input_filters/command.js	lxv
24	input_filters/remove.js	lxv
25	input_filters/line_breaks.js	lxvi
26	undo_manager.js	lxvi
27	content.js	lxviii
28	actions/type.js	lxx
29	actions/insert.js	lxxi
30	actions/remove.js	lxxiii
31	actions/format.js	lxxv
32	core_api.js	lxxvi
33	/plugins/etherpad/type.etherpad.js	lxxix
34	/plugins/etherpad/util.js	lxxx
35	/plugins/etherpad/client.js	lxxxii

36	/plugins/etherpad/content.js	lxxxv
37	/plugins/etherpad/changeset.js	lxxxvi
38	/plugins/etherpad/changes/change.js	lxxxix
39	/plugins/etherpad/changes/movement.js	lxxxix
40	/plugins/etherpad/changes/insertion.js	xc
41	/plugins/etherpad/changes/removal.js	xc i
42	/plugins/etherpad/changes/formatting.js	xc ii
43	/plugins/etherpad/changeset_serializer.js	xc iii

Bibliography

- [Berners-Lee, 1995] Berners-Lee, T. (1995). Hypertext markup language - 2.0. RFC 1866.
- [Coombs et al., 1987] Coombs, J. H., Renear, A. H., and DeRose, S. J. (1987). Markup systems and the future of scholarly text processing. *Commun. ACM*, 30(11):933–947. <http://doi.acm.org/10.1145/32206.32209>.
- [Crockford, 2008] Crockford, D. (2008). *JavaScript: The Good Parts*. O'Reilly Media / Yahoo Press.
- [Goldfarb, 1990] Goldfarb, C. F. (1990). *The SGML Handbook*. Oxford University Press, Inc., New York, NY, USA.
- [Harris, 2010] Harris, J. (2010). Google drive blog: What's different about the new google docs? <http://googledrive.blogspot.fr/2010/05/whats-different-about-new-google-docs.html>. [Online; accessed 2015-07-25].
- [ISO, 1986] ISO (1986). Standard generalized markup language (sgml). Standard Generalized Markup Language (SGML).
- [ISO, 2012] ISO (2012). Iso/iec 15445:2000. ISO/IEC 15445:2000.
- [Koszuliński, 2012] Koszuliński, P. (2012). Paste as plain text contenteditable div & textarea (word/excel...) - stack overflow. <http://stackoverflow.com/questions/11240602/paste-as-plain-text-contenteditable-div-textarea-word-excel>. [Online; accessed 2015-07-25].

- [Koszuliński, 2013] Koszuliński, P. (2013). javascript - contenteditable div vs. iframe in making a rich-text/wysiwyg editor - stack overflow. <http://stackoverflow.com/questions/10162540/contenteditable-div-vs-iframe-in-making-a-rich-text-wysiwyg-editor>. [Online; accessed 2015-07-25].
- [Microsoft, 2000a] Microsoft (2000a). contenteditable property msdn. [https://msdn.microsoft.com/en-us/library/ms533720\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms533720(v=vs.85).aspx). [Online; accessed 2015-07-25].
- [Microsoft, 2000b] Microsoft (2000b). designmode property msdn. [https://msdn.microsoft.com/en-us/library/ms537837\(VS.85\).aspx](https://msdn.microsoft.com/en-us/library/ms537837(VS.85).aspx). [Online; accessed 2015-07-25].
- [Microsoft, 2015a] Microsoft (2015a). Command identifiers (internet explorer). [https://msdn.microsoft.com/en-us/library/ms533049\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms533049(v=vs.85).aspx). [Online; accessed 2015-07-25].
- [Microsoft, 2015b] Microsoft (2015b). Methods (internet explorer). [https://msdn.microsoft.com/en-us/library/hh772123\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/hh772123(v=vs.85).aspx). [Online; accessed 2015-07-25].
- [Microsoft, 2015c] Microsoft (2015c). selection object (internet explorer). [https://msdn.microsoft.com/en-us/library/ms535869\(v=VS.85\).aspx](https://msdn.microsoft.com/en-us/library/ms535869(v=VS.85).aspx). [Online; accessed 2015-07-25].
- [Mozilla, 2003] Mozilla (2003). Rich-text editing in mozilla | mdn. https://developer.mozilla.org/en/docs/Rich-Text_Editing_in_Mozilla. [Online; accessed 2015-07-25].
- [Mozilla, 2015a] Mozilla (2015a). Content editable - web developer guide | mdn. https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Content_Editable. [Online; accessed 2015-07-25].
- [Mozilla, 2015b] Mozilla (2015b). Customevent - web api interfaces | mdn. <https://developer.mozilla.org/en/docs/Web/API/CustomEvent>. [Online; accessed 2015-07-25].

- [Mozilla, 2015c] Mozilla (2015c). `Document.execCommand()` - web api interfaces | mdn. <https://developer.mozilla.org/en-US/docs/Web/API/document/execCommand>. [Online; accessed 2015-07-25].
- [Mozilla, 2015d] Mozilla (2015d). Midas - mozilla | mdn. <https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Midas>. [Online; accessed 2015-07-25].
- [Mozilla, 2015e] Mozilla (2015e). `NodeIterator` - web api interfaces | mdn. <https://developer.mozilla.org/en-US/docs/Web/API/NodeIterator>. [Online; accessed 2015-07-25].
- [Mozilla, 2015f] Mozilla (2015f). `Selection` - web api interfaces | mdn. <https://developer.mozilla.org/en/docs/Web/HTML/Element>. [Online; accessed 2015-07-25].
- [Mozilla, 2015g] Mozilla (2015g). `Selection` - web api interfaces | mdn. <https://developer.mozilla.org/en-US/docs/Web/API/Selection>. [Online; accessed 2015-07-25].
- [Santos, 2014] Santos, N. (2014). Why contenteditable is terrible — medium engineering — medium. <https://medium.com/medium-eng/why-contenteditable-is-terrible-122d8a40e480>. [Online; accessed 2015-07-25].
- [UC, 2015] UC, U. C. (2015). Unicode bidirectional algorithm. UBA.
- [W3C, 1998] W3C (1998). Document object model (dom) level 1 specification.
- [W3C, 1999] W3C (1999). Html 4.01 specification.
- [W3C, 2012] W3C (2012). Web idl candidate recommendation.
- [W3C, 2014] W3C (2014). Html5 specification.
- [W3C, 2015] W3C (2015). Ui events w3c specification.
- [Watson, 1992] Watson, D. G. (1992). Brief history of document markup. <http://chnm.gmu.edu/digitalhistory/links/pdf/chapter3/3.19a.pdf>. [Online; accessed 2015-07-28].

[WHATWG, 2015] WHATWG (2015). Html living standard.

Part VI

Appendix

Method	Description
execCommand	Executes a command.
queryCommandEnabled	Returns whether or not a given command can currently be executed.
queryCommandIndeterm	Returns whether or not a given command is in the indeterminate state.
queryCommandState	Returns the current state of a given command.
queryCommandSupported	Returns whether or not a given command is supported by the current document's range.
queryCommandValue	Returns the value for the given command.

Table .1: HTML Editing API

Example call	Description
<code>type.caret()</code>	Returns the offset of the caret.
<code>type.caret('show')</code>	Shows the caret.
<code>type.caret('hide')</code>	Hides the caret.
<code>type.caret(10)</code>	Moves the caret to the 10th character.
<code>type.caret(10, 20)</code>	Convenience function for <code>type.select(10, 20)</code> .

Table .2: Type instance API: caret command

Example call	Description
<code>type.selection()</code>	Same as <code>type.selection('html')</code> .
<code>type.selection('text')</code>	Returns the unformatted (plain) contents of the current selection.
<code>type.selection('html')</code>	Return the currently selected HTML.
<code>type.selection(10)</code>	Convenience function for <code>type.caret(10)</code> .
<code>type.selection(10, 20)</code>	Selects characters 10 to 20.
<code>type.selection(element)</code>	Select an element.
<code>type.selection(el1, el2)</code>	Creates a selection between 2 elements.
<code>type.selection(jQuery obj.)</code>	Creates a selection between the first and last element in a jQuery object.
<code>type.selection('save')</code>	Returns an object that can be passed to <code>type.selection('restore')</code> to store and recreate a selection.
<code>type.selection('restore', sel)</code>	Takes an object returned by <code>type.selection('save')</code> as a second argument to recreate a stored selection.

Table .3: Type instance API: selection command

Example call	Description
<code>type.insert(str)</code>	Inserts formatted text at the caret's position. Will overwrite the current selection if there is one.
<code>type.insert(str, 'text')</code>	Inserts plain text at the caret's position removing all formattings from <code>str</code> . Will overwrite the current selection if there is one.
<code>type.insert(str, 10)</code>	Inserts <code>str</code> after the 10th character in the text.
<code>type.insert(str, 10, 'text')</code>	Same as <code>type.insert(str, 10)</code> but inserts unformatted text.

Table .4: Type instance API: insert command

Example call	Description
<code>type.format(htmlString)</code>	Formats the currently selected text with the markup passed as <code>htmlString</code> .
<code>type.format(htmlString, 10, 20)</code>	Formats the characters 10 to 20 in the text with the markup passed as <code>htmlString</code> .

Table .5: Type instance API: format command

Example call	Description
<code>type.remove()</code>	Deletes the currently selected text. Does nothing if there is no selection.
<code>type.remove(5)</code>	Removes 5 characters right of the caret's offset. Removes the first 5 characters of the selection if there is a text selection.
<code>type.remove(-5)</code>	Removes 5 characters left of the caret's offset. Removes the last 5 characters of the selection if there is a text selection.
<code>type.remove(10, 20)</code>	Will remove the text between the 10th and 20th character.

Table .6: Type instance API: remove command

Example call	Description
<code>type.undo()</code>	Revokes the user's last action.
<code>type.undo(5)</code>	Revokes the user's last 5 actions.
<code>type.redo()</code>	Reapplies a revoked action.
<code>type.redo(5)</code>	Reapplies 5 revoked actions.

Table .7: Type instance API: undo and redo commands

Example call	Description
<code>type.options()</code>	Returns all settings of an instance.
<code>type.options(name)</code>	Getter for a specific setting of an instance.
<code>type.options(name, values)</code>	Setter for a specific setting of an instance.
<code>type.options({name: value})</code>	Pass an object to set multiple settings of an instance.

Table .8: Type instance API: options command

Name	Licenses	Current version as of July 2015	Release of current version	First activity	Public release or release of version 1.0	SourceForge Downloads (June 2015)	GitHub Stars (June 2015)	Technology	Former names
CKEditor	GPL, LGPL, MPL	4.5.1	01/2015	03/2003	05/2003	-	1750	contentEditable	FCKEditor
TinyMCE	LGPL	4.2.10	05/2015	02/2004	04/2004	-	2369	contentEditable	
HTMLArea	BSD-style, MIT	4.0	06/2013	-	2005	84	-	designMode	
AjaxWrite	proprietary	0.9	03/2006		-	-	-	contentEditable	
Xinha	BSD	0.96.1	05/2010		-	-	-	contentEditable	HTMLArea (forked)
Epoz	Zope Public License	1.0.2	04/2013	06/2003	10/2012	-	1	contentEditable	
CB RTE	Creative Commons	3.14	09/2010	12/2003	12/2003	-	14	contentEditable	
NcEdit	MIT	0.9	06/2012	09/2011	-	-	-	contentEditable	
MediumJS	MIT	1.0.1	01/2015	08/2014	12/2014	-	3279	contentEditable	
Rich Text Editor	proprietary	8.1.0.0	-	-	-	-	-	contentEditable	
wysihtml	MIT	0.5.0-beta11	06/2015	06/2011	-	-	2561	contentEditable	wysihtml5
Quill	BSD	0.19.14	06/2015	06/2012	-	-	5305	contentEditable	
Aloha Editor	GPL, Custom	2.0.0	06/2015	03/2011	09/2011	-	2049	contentEditable	
SnapEditor	LGPL	2.0.0	01/2014	03/2012	09/2012	-	8	contentEditable	
Dijit Editor	BSD 3	1.8.10	01/2015	05/2013	05/2013	-	144	contentEditable	
MediumEditor	MIT	5.6.3	07/2015	08/2013	08/2013	-	5421	contentEditable	
Summernote	MIT	0.6.17	07/2015	06/2013	-	-	2885	contentEditable	
Redactor	proprietary	10.2.3	07/2015	2009	-	-	-	contentEditable	

Figure .1: Editors using HTML editing APIs (selection)

Name	Licenses	Current version as of July 2015	Release of current version	First activity	Public release or release of version 1.0	SourceForge Downloads (June 2015)	GitHub Stars (June 2015)	Technology	Former names
KIX (Google Docs)	proprietary	No information provided	No information provided		08/2005	-	-	JavaScript	Writely
Word online	proprietary	No information provided	No information provided		09/2009	-	-	JavaScript	
Firepad	MIT	1.1.1	05/2015	11/2013	09/2014	-	1720	JavaScript	
iCloud Pages	proprietary	No information provided	No information provided	06/2013	08/2013	-	-	JavaScript	

Figure .2: Editors not using HTML editing APIs (selection)

Caret	Development	EventApi
<ul style="list-style-type: none"> + caretEl : Element - constrainingNode : Node + moveLeft() : Caret + moveRight() : Caret + moveUp() : Caret + moveDown() : Caret + moveBy(nunChars: Number) : Caret + moveTo(node: Node, offset: Number) : Caret + destroy() : Caret + getOffset() : Number + setOffset(offset: Number) : Caret + getNodeOffset() : Number + getNode() : Node + setOffset() : Caret - moveEToOffset() : Caret - moveETo() : Caret - setElHeight() : Caret - blink() : Caret - hide() : Caret - resetBlink() : Caret - getRectAOffset() : Object - getPositionsFromRange() : Object - getScrollPosition() : Object - createRange() : Range - createElement() : Element 	<ul style="list-style-type: none"> + log (message: String) : Development + debug (message: String) : Development <ul style="list-style-type: none"> - containerId : String - singleTag : RegExp <ul style="list-style-type: none"> + addElement (tagName: String, className: String) : Element + removeElement (el: Node) : DomUtilities + removeVisible (node: Node, constrainingNode: Node) : Element + removeTag (el: Node, tag: String, deep: Boolean) : Element + parseHTML (htmlString: String) : Array + wrap (tag: String, elms: Array) : Element + unwrap (el: Element) : DomUtilities + moveAfter (reference: Node, elms: Array) : DomUtilities + parentAfter (node: selector: String, constrainingNode: Node) : Element + matches (el: Node, selector: String) : Boolean + getElementsContainer () : Element + containsButtset (container: Node, node: Node) : Boolean + isNode (obj: Object) : Boolean + isVisible (el: Node) : Boolean + order (a: Node, b: Node) : Number 	<ul style="list-style-type: none"> + on (eventName: String, cb: Function) : EventApi + off (eventName: String, cb: Function) : EventApi + trigger (eventName: String, params: Object) : EventApi + on (eventName: String, cb: Function) : EventApi + off (eventName: String, cb: Function) : EventApi + trigger (eventName: String, params: Object) : EventApi
DomUtilities	DomWalker	Input
<ul style="list-style-type: none"> + addElement (tagName: String, className: String) : Element + removeElement (el: Node) : DomUtilities + removeVisible (node: Node, constrainingNode: Node) : Element + removeTag (el: Node, tag: String, deep: Boolean) : Element + parseHTML (htmlString: String) : Array + wrap (tag: String, elms: Array) : Element + unwrap (el: Element) : DomUtilities + moveAfter (reference: Node, elms: Array) : DomUtilities + parentAfter (node: selector: String, constrainingNode: Node) : Element + matches (el: Node, selector: String) : Boolean + getElementsContainer () : Element + containsButtset (container: Node, node: Node) : Boolean + isNode (obj: Object) : Boolean + isVisible (el: Node) : Boolean + order (a: Node, b: Node) : Number 	<ul style="list-style-type: none"> - filterFunctions : Object + next (returnMe: Boolean) : Node + prev (returnMe: Boolean) : Node + first () : Node + last () : Node + loadOptions (options) : Object + next (returnMe: Boolean) : Node + prevNext (returnMe: Boolean) : Node + prev (returnMe: Boolean) : Node + prevNextPrev (returnMe: Boolean) : Node + first () : Node + last () : Node + setNode (node: Node) : DomWalker + options (options: Object) : DomWalker + getNode () : Node - badFilter (filter) : Function - nextNode (node, options, returnMe) : Node - prevNode (node, options, returnMe) : Node - isTextNode (node) : Boolean - isTextNodeWithContents (node) : Boolean - resemblesText (node) : Boolean - isVisible (node) : Boolean - setNodeIfNotNull (node: Node) : DomWalker 	<ul style="list-style-type: none"> - type: Type - content: Content - writer: Writer - caret: Caret - selection: Selection - el: Element - filters: Array <ul style="list-style-type: none"> + addFilter (name: String, filter: Object) : Input + removeFilter (name: String) : Input + getContents () : Content - loadFilters () : Input - bindEvents () : Input - bindKeyDownEvents () : Input - bindInputEvents () : Input - processFilterPipeline (e: KeyboardEvent) : InputEvent - processFilter (filter, InputEvent) : InputEvent - onInput () : Input - moveCaretToMousePosition(x: Number, y: Number) : Input - focusInput (sync: Boolean) : Input - createElement () : Element
Environment	Format	OOP
<ul style="list-style-type: none"> + mac : Boolean 	<ul style="list-style-type: none"> - inline Tags : Array - blockTags : Array + format (tag: String, typeRange: Range, params: Object) : Array + removeFormat (tag: String, range: Range) : Format + inline (tag: String, typeRange: Range, params: Object) : Array + insertLine (tag: String, startNode: Node, endNode: Node, params: Object) : Format + removeLine (enclosing tag: Node, typeRange: Range) : Node - getStartNode (tag: String, typeRange: Range) : Node - getEndNode (tag: String, typeRange: Range) : Node - handleIf (tag: String) : Function - noop () : Function 	<ul style="list-style-type: none"> - constructor: Function, parentConstructor: Function) : Function
Content	Type	
<ul style="list-style-type: none"> - sourceId : Object - undoManager : UndoManager - writer : Writer - formatter : Formatter - root : Element - type : Type + insert (textNode: Node, offset: Number, content: String) : Content + remove (range: Range, numCharacters: Number) : Content + format (tag, range, end) : Content + removeFormat (tag: String, range: Range, end: Offset) : Content + getSourceId () : String + getRoot () : Element - createUniqueSourceId () : String 	<ul style="list-style-type: none"> - root : Element - options : Object - writer : Writer - formatter : Formatter - undoManager : UndoManager - caret : Caret - selection : Selection - input : Input + createDomWalker (node: Node, options: Object) : DomWalker + getRoot () : Root + getCaret () : Caret + getSelection () : Selection + getUndoManager () : UndoManager + getWriter () : Writer + getFormatter () : Formatter + getInput () : Input 	

Figure .3: Classes of the library 1 /3

CaretFilter
+ keys: Object - caret: Caret
+ moveLeft (e: InputEvent) : void + moveUp (e: InputEvent) : void + moveRight (e: InputEvent) : void + moveDown (e: InputEvent) : void

CommandFilter
+ keys: Object + tags: Object - selection: Selection - content: Content
+ command (e: InputEvent) : void

LineBreaksFilter
+ keys: Object - writer: Writer - caret: Caret
+ insertLineBreak (e: InputEvent) : void

RemoveFilter
+ keys: Object - root: Element - content: Content - caret: Caret - selection: Selection
+ remove (e: InputEvent) : void

UndoFilter
+ keys: Object - undoManager: UndoManager - sourceId: Content
+ undoRedo (e: InputEvent) : void

Figure .5: Classes of the library 3 / 3

Etherpad		EtherpadChangeset		Change	
<ul style="list-style-type: none"> -type: Type -options: Object -defaultOptions: Object -client: EtherpadClient -content: Content 		<ul style="list-style-type: none"> -stack: Array -changesetRegex: RegExp 		<ul style="list-style-type: none"> +op: String 	
<ul style="list-style-type: none"> +prepareOptions (options: Object, etherpadOpts: Object, server: String): Object +options (options, value): Object +getType (): Type +getClient (): EtherpadClient +initEditor (contents, apool): Etherpad +initChangeset (contents): String 		<ul style="list-style-type: none"> +fromString (str: String, apool: Object, base: String): EtherpadChangeset +getString (base: String): String +apply (content: Content, localCaret: Caret): EtherpadChangeset +addString (str: String, apool: Object, base: String): EtherpadChangeset +getStack (): Array +getNilIndices (str: String): Array +addMatchToStack (offset: Number, charbank: String, match: Object, apool: Object, nllIndices: Array): EtherpadChangeset -createFromMatch (offset: Number, charbank: String, match: Object, apool: Object): ChangesetChange -operationOrMovement (offset: Number, charbank: String, match: Object, apool: Object): ChangesetChange -mergeOrPush (change: ChangesetChange): EtherpadChangeset -parseMatch (match: Object): Object -getAttributesFromMatch (match: Object, apool: Object): Object -getCharbank (str: String): String 		<ul style="list-style-type: none"> +fromMatch (match: Object): Change +apply (content, localCaret): Change +mergeable (that: Change): Boolean +merge (that: Change): Change +getCharbank (): String +getOperation (): String +getLength (): Number 	
EtherpadClient		ChangesetSerializer		RemovalChange	
<ul style="list-style-type: none"> -etherpad: Etherpad; -msgHandlers: Object -defaultUrl: String 		<ul style="list-style-type: none"> -operations: Array 		<ul style="list-style-type: none"> +start: Number +length: Number +end: Number 	
<ul style="list-style-type: none"> +connect (): EtherpadClient +onInit (handler: Function): EtherpadClient +registerMessageHandler (msg: String, handler: Function): EtherpadClient +unregisterMessageHandler (msg: String, handler: Function): EtherpadClient -handleMessage (response: Object): EtherpadClient -init (data: Object): EtherpadClient -sendClientReady(): EtherpadClient -socketOptions (): Object -url (): String 		<ul style="list-style-type: none"> +getString (base: String): String -baseLengthString (base: String): String -lengthChangeString (): String -operationString (operation: Change, prev: Change): String -charbankString (): String -operationCharbankString (operation: Change): String -offsetString (operation: Change, prev: Change): String -lengthFor (base: String): Number -countLengthChange (): Number -getOperations (changeset: String): Array -compareOperations (a: Change, b: Change): Number 		<ul style="list-style-type: none"> +fromMatch (offset: Number, match: Object): RemovalChange 	
EtherpadContent		FormattingChange		FormattingChange	
<ul style="list-style-type: none"> -client: EtherpadClient -localCaret: Caret -typeContent: Content -root: Element 		<ul style="list-style-type: none"> +start: Number +length: Number +end: Number +command: String -tagMap: Object 		<ul style="list-style-type: none"> +start: Number +length: Number +end: Number +command: String -tagMap: Object 	
<ul style="list-style-type: none"> +updateContent (data: Object): EtherpadContent +applyChangeset (changesetString: String, apool: Object): EtherpadContent 		<ul style="list-style-type: none"> +fromAttrs (attrs: Array, offset: Number, match: Object): FormattingChange 		<ul style="list-style-type: none"> +fromAttrs (attrs: Array, offset: Number, match: Object): FormattingChange 	
EtherpadUtil		MovementChange		MovementChange	
<ul style="list-style-type: none"> +n2br (str: String) +getRandomToken (): String 		<ul style="list-style-type: none"> +delta: Number +absolute: Number 		<ul style="list-style-type: none"> +delta: Number +absolute: Number 	
				<ul style="list-style-type: none"> +fromOffsetObject (offset: Object, match: Object): FormattingChange 	

Figure .6: Classes of the Etherpad Extension

Part VII

Source Code

Source Code

```

1  'use strict';
2
3  /**
4   * Creates a new Type editor and
5   * sets up the core
6   * modules used for WYSIWYG editing.
7   * The core
8   * class only holds methods for
9   * setting and retrieving
10  * options as well getters and
11  * setters for instances
12  * of core modules.
13
14  * @param {{}|Element} options -
15  *   Either pass
16  *   an associative array with
17  *   options for this
18  *   editor or the root element
19  *   that should be
20  *   used to modify its contents
21  *   for WYSIWYG
22  *   editing
23  * @param {Element} options.el The
24  *   root element
25  *   that should be used to modify
26  *   its contents
27  *   for WYSIWYG editing
28  * @constructor
29
30  function Type(options) {
31
32     // Allow passing an element as
33     // only parameter
34     if (Type.DomUtilities.isNode(
35         options)) {
36         options = { el: options };
37     }
38
39     // If no element has been passed,
40     // interrupt
41     if (!options.el) {
42         throw new Error('You must
43             provide an element as root
44             node for the editor\'s
45             TypeContents.');
```

```

46     }
47
48     // Set settings for this editor
49     this._root = null;
50     this.options(options);
51
52     // Set up core modules
53     this._writer = new Type.
54         Writer(this);
55     this._formatter = new Type.
56         Formatter(this);
57     this._undoManager = new Type.
58         UndoManager(this);
59     //this._content = new Type.
60         Content(this);
61     this._caret = new Type.
62         Caret(this);
63     this._selection = new Type.
64         Selection(this);
65     this._input = new Type.
66         Input(this);
67
68     // Trigger events
69     Type.trigger('ready', this);
70
71     (function () {
72
73         /**
74          * Allows fast detection if an
75          * object is a Type Editor
76          * instance (or class)
77          * @type {boolean}
78          */
79         this.typeEditor = true;
80
81         /**
82          * Holds the default options for
83          * every editor. These options
84          * will be extended by the options
85          * passed to each instance
86          * on instantiation.
87          */
88     })
89
90     // Set settings for this editor
91     this._root = null;
92     this.options(options);
93
94     // Set up core modules
95     this._writer = new Type.
96         Writer(this);
97     this._formatter = new Type.
98         Formatter(this);
99     this._undoManager = new Type.
100         UndoManager(this);
101     //this._content = new Type.
102         Content(this);
103     this._caret = new Type.
104         Caret(this);
105     this._selection = new Type.
106         Selection(this);
107     this._input = new Type.
108         Input(this);
109
110     // Trigger events
111     Type.trigger('ready', this);
112
113     (function () {
114
115         /**
116          * Allows fast detection if an
117          * object is a Type Editor
118          * instance (or class)
119          * @type {boolean}
120          */
121         this.typeEditor = true;
122
123         /**
124          * Holds the default options for
125          * every editor. These options
126          * will be extended by the options
127          * passed to each instance
128          * on instantiation.
129          */
130     })
131
132     // Set settings for this editor
133     this._root = null;
134     this.options(options);
135
136     // Set up core modules
137     this._writer = new Type.
138         Writer(this);
139     this._formatter = new Type.
140         Formatter(this);
141     this._undoManager = new Type.
142         UndoManager(this);
143     //this._content = new Type.
144         Content(this);
145     this._caret = new Type.
146         Caret(this);
147     this._selection = new Type.
148         Selection(this);
149     this._input = new Type.
150         Input(this);
151
152     // Trigger events
153     Type.trigger('ready', this);
154
155     (function () {
156
157         /**
158          * Allows fast detection if an
159          * object is a Type Editor
160          * instance (or class)
161          * @type {boolean}
162          */
163         this.typeEditor = true;
164
165         /**
166          * Holds the default options for
167          * every editor. These options
168          * will be extended by the options
169          * passed to each instance
170          * on instantiation.
171          */
172     })
173
174     // Set settings for this editor
175     this._root = null;
176     this.options(options);
177
178     // Set up core modules
179     this._writer = new Type.
180         Writer(this);
181     this._formatter = new Type.
182         Formatter(this);
183     this._undoManager = new Type.
184         UndoManager(this);
185     //this._content = new Type.
186         Content(this);
187     this._caret = new Type.
188         Caret(this);
189     this._selection = new Type.
190         Selection(this);
191     this._input = new Type.
192         Input(this);
193
194     // Trigger events
195     Type.trigger('ready', this);
196
197     (function () {
198
199         /**
200          * Allows fast detection if an
201          * object is a Type Editor
202          * instance (or class)
203          * @type {boolean}
204          */
205         this.typeEditor = true;
206
207         /**
208          * Holds the default options for
209          * every editor. These options
210          * will be extended by the options
211          * passed to each instance
212          * on instantiation.
213          */
214     })
215
216     // Set settings for this editor
217     this._root = null;
218     this.options(options);
219
220     // Set up core modules
221     this._writer = new Type.
222         Writer(this);
223     this._formatter = new Type.
224         Formatter(this);
225     this._undoManager = new Type.
226         UndoManager(this);
227     //this._content = new Type.
228         Content(this);
229     this._caret = new Type.
230         Caret(this);
231     this._selection = new Type.
232         Selection(this);
233     this._input = new Type.
234         Input(this);
235
236     // Trigger events
237     Type.trigger('ready', this);
238
239     (function () {
240
241         /**
242          * Allows fast detection if an
243          * object is a Type Editor
244          * instance (or class)
245          * @type {boolean}
246          */
247         this.typeEditor = true;
248
249         /**
250          * Holds the default options for
251          * every editor. These options
252          * will be extended by the options
253          * passed to each instance
254          * on instantiation.
255          */
256     })
257
258     // Set settings for this editor
259     this._root = null;
260     this.options(options);
261
262     // Set up core modules
263     this._writer = new Type.
264         Writer(this);
265     this._formatter = new Type.
266         Formatter(this);
267     this._undoManager = new Type.
268         UndoManager(this);
269     //this._content = new Type.
270         Content(this);
271     this._caret = new Type.
272         Caret(this);
273     this._selection = new Type.
274         Selection(this);
275     this._input = new Type.
276         Input(this);
277
278     // Trigger events
279     Type.trigger('ready', this);
280
281     (function () {
282
283         /**
284          * Allows fast detection if an
285          * object is a Type Editor
286          * instance (or class)
287          * @type {boolean}
288          */
289         this.typeEditor = true;
290
291         /**
292          * Holds the default options for
293          * every editor. These options
294          * will be extended by the options
295          * passed to each instance
296          * on instantiation.
297          */
298     })
299
300     // Set settings for this editor
301     this._root = null;
302     this.options(options);
303
304     // Set up core modules
305     this._writer = new Type.
306         Writer(this);
307     this._formatter = new Type.
308         Formatter(this);
309     this._undoManager = new Type.
310         UndoManager(this);
311     //this._content = new Type.
312         Content(this);
313     this._caret = new Type.
314         Caret(this);
315     this._selection = new Type.
316         Selection(this);
317     this._input = new Type.
318         Input(this);
319
320     // Trigger events
321     Type.trigger('ready', this);
322
323     (function () {
324
325         /**
326          * Allows fast detection if an
327          * object is a Type Editor
328          * instance (or class)
329          * @type {boolean}
330          */
331         this.typeEditor = true;
332
333         /**
334          * Holds the default options for
335          * every editor. These options
336          * will be extended by the options
337          * passed to each instance
338          * on instantiation.
339          */
340     })
341
342     // Set settings for this editor
343     this._root = null;
344     this.options(options);
345
346     // Set up core modules
347     this._writer = new Type.
348         Writer(this);
349     this._formatter = new Type.
350         Formatter(this);
351     this._undoManager = new Type.
352         UndoManager(this);
353     //this._content = new Type.
354         Content(this);
355     this._caret = new Type.
356         Caret(this);
357     this._selection = new Type.
358         Selection(this);
359     this._input = new Type.
360         Input(this);
361
362     // Trigger events
363     Type.trigger('ready', this);
364
365     (function () {
366
367         /**
368          * Allows fast detection if an
369          * object is a Type Editor
370          * instance (or class)
371          * @type {boolean}
372          */
373         this.typeEditor = true;
374
375         /**
376          * Holds the default options for
377          * every editor. These options
378          * will be extended by the options
379          * passed to each instance
380          * on instantiation.
381          */
382     })
383
384     // Set settings for this editor
385     this._root = null;
386     this.options(options);
387
388     // Set up core modules
389     this._writer = new Type.
390         Writer(this);
391     this._formatter = new Type.
392         Formatter(this);
393     this._undoManager = new Type.
394         UndoManager(this);
395     //this._content = new Type.
396         Content(this);
397     this._caret = new Type.
398         Caret(this);
399     this._selection = new Type.
400         Selection(this);
401     this._input = new Type.
402         Input(this);
403
404     // Trigger events
405     Type.trigger('ready', this);
406
407     (function () {
408
409         /**
410          * Allows fast detection if an
411          * object is a Type Editor
412          * instance (or class)
413          * @type {boolean}
414          */
415         this.typeEditor = true;
416
417         /**
418          * Holds the default options for
419          * every editor. These options
420          * will be extended by the options
421          * passed to each instance
422          * on instantiation.
423          */
424     })
425
426     // Set settings for this editor
427     this._root = null;
428     this.options(options);
429
430     // Set up core modules
431     this._writer = new Type.
432         Writer(this);
433     this._formatter = new Type.
434         Formatter(this);
435     this._undoManager = new Type.
436         UndoManager(this);
437     //this._content = new Type.
438         Content(this);
439     this._caret = new Type.
440         Caret(this);
441     this._selection = new Type.
442         Selection(this);
443     this._input = new Type.
444         Input(this);
445
446     // Trigger events
447     Type.trigger('ready', this);
448
449     (function () {
450
451         /**
452          * Allows fast detection if an
453          * object is a Type Editor
454          * instance (or class)
455          * @type {boolean}
456          */
457         this.typeEditor = true;
458
459         /**
460          * Holds the default options for
461          * every editor. These options
462          * will be extended by the options
463          * passed to each instance
464          * on instantiation.
465          */
466     })
467
468     // Set settings for this editor
469     this._root = null;
470     this.options(options);
471
472     // Set up core modules
473     this._writer = new Type.
474         Writer(this);
475     this._formatter = new Type.
476         Formatter(this);
477     this._undoManager = new Type.
478         UndoManager(this);
479     //this._content = new Type.
480         Content(this);
481     this._caret = new Type.
482         Caret(this);
483     this._selection = new Type.
484         Selection(this);
485     this._input = new Type.
486         Input(this);
487
488     // Trigger events
489     Type.trigger('ready', this);
490
491     (function () {
492
493         /**
494          * Allows fast detection if an
495          * object is a Type Editor
496          * instance (or class)
497          * @type {boolean}
498          */
499         this.typeEditor = true;
500
501         /**
502          * Holds the default options for
503          * every editor. These options
504          * will be extended by the options
505          * passed to each instance
506          * on instantiation.
507          */
508     })
509
510     // Set settings for this editor
511     this._root = null;
512     this.options(options);
513
514     // Set up core modules
515     this._writer = new Type.
516         Writer(this);
517     this._formatter = new Type.
518         Formatter(this);
519     this._undoManager = new Type.
520         UndoManager(this);
521     //this._content = new Type.
522         Content(this);
523     this._caret = new Type.
524         Caret(this);
525     this._selection = new Type.
526         Selection(this);
527     this._input = new Type.
528         Input(this);
529
530     // Trigger events
531     Type.trigger('ready', this);
532
533     (function () {
534
535         /**
536          * Allows fast detection if an
537          * object is a Type Editor
538          * instance (or class)
539          * @type {boolean}
540          */
541         this.typeEditor = true;
542
543         /**
544          * Holds the default options for
545          * every editor. These options
546          * will be extended by the options
547          * passed to each instance
548          * on instantiation.
549          */
550     })
551
552     // Set settings for this editor
553     this._root = null;
554     this.options(options);
555
556     // Set up core modules
557     this._writer = new Type.
558         Writer(this);
559     this._formatter = new Type.
560         Formatter(this);
561     this._undoManager = new Type.
562         UndoManager(this);
563     //this._content = new Type.
564         Content(this);
565     this._caret = new Type.
566         Caret(this);
567     this._selection = new Type.
568         Selection(this);
569     this._input = new Type.
570         Input(this);
571
572     // Trigger events
573     Type.trigger('ready', this);
574
575     (function () {
576
577         /**
578          * Allows fast detection if an
579          * object is a Type Editor
580          * instance (or class)
581          * @type {boolean}
582          */
583         this.typeEditor = true;
584
585         /**
586          * Holds the default options for
587          * every editor. These options
588          * will be extended by the options
589          * passed to each instance
590          * on instantiation.
591          */
592     })
593
594     // Set settings for this editor
595     this._root = null;
596     this.options(options);
597
598     // Set up core modules
599     this._writer = new Type.
600         Writer(this);
601     this._formatter = new Type.
602         Formatter(this);
603     this._undoManager = new Type.
604         UndoManager(this);
605     //this._content = new Type.
606         Content(this);
607     this._caret = new Type.
608         Caret(this);
609     this._selection = new Type.
610         Selection(this);
611     this._input = new Type.
612         Input(this);
613
614     // Trigger events
615     Type.trigger('ready', this);
616
617     (function () {
618
619         /**
620          * Allows fast detection if an
621          * object is a Type Editor
622          * instance (or class)
623          * @type {boolean}
624          */
625         this.typeEditor = true;
626
627         /**
628          * Holds the default options for
629          * every editor. These options
630          * will be extended by the options
631          * passed to each instance
632          * on instantiation.
633          */
634     })
635
636     // Set settings for this editor
637     this._root = null;
638     this.options(options);
639
640     // Set up core modules
641     this._writer = new Type.
642         Writer(this);
643     this._formatter = new Type.
644         Formatter(this);
645     this._undoManager = new Type.
646         UndoManager(this);
647     //this._content = new Type.
648         Content(this);
649     this._caret = new Type.
650         Caret(this);
651     this._selection = new Type.
652         Selection(this);
653     this._input = new Type.
654         Input(this);
655
656     // Trigger events
657     Type.trigger('ready', this);
658
659     (function () {
660
661         /**
662          * Allows fast detection if an
663          * object is a Type Editor
664          * instance (or class)
665          * @type {boolean}
666          */
667         this.typeEditor = true;
668
669         /**
670          * Holds the default options for
671          * every editor. These options
672          * will be extended by the options
673          * passed to each instance
674          * on instantiation.
675          */
676     })
677
678     // Set settings for this editor
679     this._root = null;
680     this.options(options);
681
682     // Set up core modules
683     this._writer = new Type.
684         Writer(this);
685     this._formatter = new Type.
686         Formatter(this);
687     this._undoManager = new Type.
688         UndoManager(this);
689     //this._content = new Type.
690         Content(this);
691     this._caret = new Type.
692         Caret(this);
693     this._selection = new Type.
694         Selection(this);
695     this._input = new Type.
696         Input(this);
697
698     // Trigger events
699     Type.trigger('ready', this);
700
701     (function () {
702
703         /**
704          * Allows fast detection if an
705          * object is a Type Editor
706          * instance (or class)
707          * @type {boolean}
708          */
709         this.typeEditor = true;
710
711         /**
712          * Holds the default options for
713          * every editor. These options
714          * will be extended by the options
715          * passed to each instance
716          * on instantiation.
717          */
718     })
719
720     // Set settings for this editor
721     this._root = null;
722     this.options(options);
723
724     // Set up core modules
725     this._writer = new Type.
726         Writer(this);
727     this._formatter = new Type.
728         Formatter(this);
729     this._undoManager = new Type.
730         UndoManager(this);
731     //this._content = new Type.
732         Content(this);
733     this._caret = new Type.
734         Caret(this);
735     this._selection = new Type.
736         Selection(this);
737     this._input = new Type.
738         Input(this);
739
740     // Trigger events
741     Type.trigger('ready', this);
742
743     (function () {
744
745         /**
746          * Allows fast detection if an
747          * object is a Type Editor
748          * instance (or class)
749          * @type {boolean}
750          */
751         this.typeEditor = true;
752
753         /**
754          * Holds the default options for
755          * every editor. These options
756          * will be extended by the options
757          * passed to each instance
758          * on instantiation.
759          */
760     })
761
762     // Set settings for this editor
763     this._root = null;
764     this.options(options);
765
766     // Set up core modules
767     this._writer = new Type.
768         Writer(this);
769     this._formatter = new Type.
770         Formatter(this);
771     this._undoManager = new Type.
772         UndoManager(this);
773     //this._content = new Type.
774         Content(this);
775     this._caret = new Type.
776         Caret(this);
777     this._selection = new Type.
778         Selection(this);
779     this._input = new Type.
780         Input(this);
781
782     // Trigger events
783     Type.trigger('ready', this);
784
785     (function () {
786
787         /**
788          * Allows fast detection if an
789          * object is a Type Editor
790          * instance (or class)
791          * @type {boolean}
792          */
793         this.typeEditor = true;
794
795         /**
796          * Holds the default options for
797          * every editor. These options
798          * will be extended by the options
799          * passed to each instance
800          * on instantiation.
801          */
802     })
803
804     // Set settings for this editor
805     this._root = null;
806     this.options(options);
807
808     // Set up core modules
809     this._writer = new Type.
810         Writer(this);
811     this._formatter = new Type.
812         Formatter(this);
813     this._undoManager = new Type.
814         UndoManager(this);
815     //this._content = new Type.
816         Content(this);
817     this._caret = new Type.
818         Caret(this);
819     this._selection = new Type.
820         Selection(this);
821     this._input = new Type.
822         Input(this);
823
824     // Trigger events
825     Type.trigger('ready', this);
826
827     (function () {
828
829         /**
830          * Allows fast detection if an
831          * object is a Type Editor
832          * instance (or class)
833          * @type {boolean}
834          */
835         this.typeEditor = true;
836
837         /**
838          * Holds the default options for
839          * every editor. These options
840          * will be extended by the options
841          * passed to each instance
842          * on instantiation.
843          */
844     })
845
846     // Set settings for this editor
847     this._root = null;
848     this.options(options);
849
850     // Set up core modules
851     this._writer = new Type.
852         Writer(this);
853     this._formatter = new Type.
854         Formatter(this);
855     this._undoManager = new Type.
856         UndoManager(this);
857     //this._content = new Type.
858         Content(this);
859     this._caret = new Type.
860         Caret(this);
861     this._selection = new Type.
862         Selection(this);
863     this._input = new Type.
864         Input(this);
865
866     // Trigger events
867     Type.trigger('ready', this);
868
869     (function () {
870
871         /**
872          * Allows fast detection if an
873          * object is a Type Editor
874          * instance (or class)
875          * @type {boolean}
876          */
877         this.typeEditor = true;
878
879         /**
880          * Holds the default options for
881          * every editor. These options
882          * will be extended by the options
883          * passed to each instance
884          * on instantiation.
885          */
886     })
887
888     // Set settings for this editor
889     this._root = null;
890     this.options(options);
891
892     // Set up core modules
893     this._writer = new Type.
894         Writer(this);
895     this._formatter = new Type.
896         Formatter(this);
897     this._undoManager = new Type.
898         UndoManager(this);
899     //this._content = new Type.
900         Content(this);
901     this._caret = new Type.
902         Caret(this);
903     this._selection = new Type.
904         Selection(this);
905     this._input = new Type.
906         Input(this);
907
908     // Trigger events
909     Type.trigger('ready', this);
910
911     (function () {
912
913         /**
914          * Allows fast detection if an
915          * object is a Type Editor
916          * instance (or class)
917          * @type {boolean}
918          */
919         this.typeEditor = true;
920
921         /**
922          * Holds the default options for
923          * every editor. These options
924          * will be extended by the options
925          * passed to each instance
926          * on instantiation.
927          */
928     })
929
930     // Set settings for this editor
931     this._root = null;
932     this.options(options);
933
934     // Set up core modules
935     this._writer = new Type.
936         Writer(this);
937     this._formatter = new Type.
938         Formatter(this);
939     this._undoManager = new Type.
940         UndoManager(this);
941     //this._content = new Type.
942         Content(this);
943     this._caret = new Type.
944         Caret(this);
945     this._selection = new Type.
946         Selection(this);
947     this._input = new Type.
948         Input(this);
949
950     // Trigger events
951     Type.trigger('ready', this);
952
953     (function () {
954
955         /**
956          * Allows fast detection if an
957          * object is a Type Editor
958          * instance (or class)
959          * @type {boolean}
960          */
961         this.typeEditor = true;
962
963         /**
964          * Holds the default options for
965          * every editor. These options
966          * will be extended by the options
967          * passed to each instance
968          * on instantiation.
969          */
970     })
971
972     // Set settings for this editor
973     this._root = null;
974     this.options(options);
975
976     // Set up core modules
977     this._writer = new Type.
978         Writer(this);
979     this._formatter = new Type.
980         Formatter(this);
981     this._undoManager = new Type.
982         UndoManager(this);
983     //this._content = new Type.
984         Content(this);
985     this._caret = new Type.
986         Caret(this);
987     this._selection = new Type.
988         Selection(this);
989     this._input = new Type.
990         Input(this);
991
992     // Trigger events
993     Type.trigger('ready', this);
994
995     (function () {
996
997         /**
998          * Allows fast detection if an
999          * object is a Type Editor
1000          * instance (or class)
1001          * @type {boolean}
1002          */
1003         this.typeEditor = true;
1004
1005         /**
1006          * Holds the default options for
1007          * every editor. These options
1008          * will be extended by the options
1009          * passed to each instance
1010          * on instantiation.
1011          */
1012     })
1013
1014     // Set settings for this editor
1015     this._root = null;
1016     this.options(options);
1017
1018     // Set up core modules
1019     this._writer = new Type.
1020         Writer(this);
1021     this._formatter = new Type.
1022         Formatter(this);
1023     this._undoManager = new Type.
1024         UndoManager(this);
1025     //this._content = new Type.
1026         Content(this);
1027     this._caret = new Type.
1028         Caret(this);
1029     this._selection = new Type.
1030         Selection(this);
1031     this._input = new Type.
1032         Input(this);
1033
1034     // Trigger events
1035     Type.trigger('ready', this);
1036
1037     (function () {
1038
1039         /**
1040          * Allows fast detection if an
1041          * object is a Type Editor
1042          * instance (or class)
1043          * @type {boolean}
1044          */
1045         this.typeEditor = true;
1046
1047         /**
1048          * Holds the default options for
1049          * every editor. These options
1050          * will be extended by the options
1051          * passed to each instance
1052          * on instantiation.
1053          */
1054     })
1055
1056     // Set settings for this editor
1057     this._root = null;
1058     this.options(options);
1059
1060     // Set up core modules
1061     this._writer = new Type.
1062         Writer(this);
1063     this._formatter = new Type.
1064         Formatter(this);
1065     this._undoManager = new Type.
1066         UndoManager(this);
1067     //this._content = new Type.
1068         Content(this);
1069     this._caret = new Type.
1070         Caret(this);
1071     this._selection = new Type.
1072         Selection(this);
1073     this._input = new Type.
1074         Input(this);
1075
1076     // Trigger events
1077     Type.trigger('ready', this);
1078
1079     (function () {
1080
1081         /**
1082          * Allows fast detection if an
1083          * object is a Type Editor
1084          * instance (or class)
1085          * @type {boolean}
1086          */
1087         this.typeEditor = true;
1088
1089         /**
1090          * Holds the default options for
1091          * every editor. These options
1092          * will be extended by the options
1093          * passed to each instance
1094          * on instantiation.
1095          */
1096     })
1097
1098     // Set settings for this editor
1099     this._root = null;
1100     this.options(options);
1101
1102     // Set up core modules
1103     this._writer = new Type.
1104         Writer(this);
1105     this._formatter = new Type.
1106         Formatter(this);
1107     this._undoManager = new Type.
1108         UndoManager(this);
1109     //this._content = new Type.
1110         Content(this);
1111     this._caret = new Type.
1112         Caret(this);
1113     this._selection = new Type.
1114         Selection(this);
1115     this._input = new Type.
1116         Input(this);
1117
1118     // Trigger events
1119     Type.trigger('ready', this);
1120
1121     (function () {
1122
1123         /**
1124          * Allows fast detection if an
1125          * object is a Type Editor
1126          * instance (or class)
1127          * @type {boolean}
1128          */
1129         this.typeEditor = true;
1130
1131         /**
1132          * Holds the default options for
1133          * every editor. These options
1134          * will be extended by the options
1135          * passed to each instance
1136          * on instantiation.
1137          */
1138     })
1139
1140     // Set settings for this editor
1141     this._root = null;
1142     this.options(options);
1143
1144     // Set up core modules
1145     this._writer = new Type.
1146         Writer(this);
1147     this._formatter = new Type.
1148         Formatter(this);
1149     this._undoManager = new Type.
1150         UndoManager(this);
1151     //this._content = new Type.
1152         Content(this);
1153     this._caret = new Type.
1154         Caret(this);
1155     this._selection = new Type.
1156         Selection(this);
1157     this._input = new Type.
1158         Input(this);
1159
1160     // Trigger events
1161     Type.trigger('ready', this);
1162
1163     (function () {
1164
1165         /**
1166          * Allows fast detection if an
1167          * object is a Type Editor
1168          * instance (or class)
1169          * @type {boolean}
1170          */
1171         this.typeEditor = true;
1172
1173         /**
1174          * Holds the default options for
1175          * every editor. These options
1176          * will be extended by the options
1177          * passed to each instance
1178          * on instantiation.
1179          */
1180     })
1181
1182     // Set settings for this editor
1183     this._root = null;
1184     this.options(options);
1185
1186     // Set up core modules
1187     this._writer = new Type.
1188         Writer(this);
1189     this._formatter = new Type.
1190         Formatter(this);
1191     this._undoManager = new Type.
1192         UndoManager(this);
1193     //this._content = new Type.
1194         Content(this);
1195     this._caret = new Type.
1196         Caret(this);
1197     this._selection = new Type.
1198         Selection(this);
1199     this._input = new Type.
1200         Input(this);
1201
1202     // Trigger events
1203     Type.trigger('ready', this);
1204
1205     (function () {
1206
1207         /**
1208          * Allows fast detection if an
1209          * object is a Type Editor
1210          * instance (or class)
1211          * @type {boolean}
1212          */
1213         this.typeEditor = true;
1214
1215         /**
1216          * Holds the default options for
1217          * every editor. These options
1218          * will be extended by the options
1219          * passed to each instance
1220          * on instantiation.
1221          */
1222     })
1223
1224     // Set settings for this editor
1225     this._root = null;
1226     this.options(options);
1227
1228     // Set up core modules
1229     this._writer = new Type.
1230         Writer(this);
1231     this._formatter = new Type.
1232         Formatter(this);
1233     this._undoManager = new Type.
1234         UndoManager(this);
1235     //this._content = new Type.
1236         Content(this);
1237     this._caret = new Type.
1238         Caret(this);
1239     this._selection = new Type.
1240         Selection(this);
1241     this._input = new Type.
1242         Input(this);
1243
1244     // Trigger events
1245     Type.trigger('ready', this);
1246
1247     (function () {
1248
1249         /**
1250          * Allows fast detection if an
1251          * object is a Type Editor
1252          * instance (or class)
1253          * @type {boolean}
1254          */
1255         this.typeEditor = true;
1256
1257         /**
1258          * Holds the default options for
1259          * every editor. These options
1260          * will be extended by the options
1261          * passed to each instance
1262          * on instantiation.
1263          */
1264     })
1265
1266     // Set settings for this editor
1267     this._root = null;
1268     this.options(options);
1269
1270     // Set up core modules
1271     this._writer = new Type.
1272         Writer(this);
1273     this._formatter = new Type.
1274         Formatter(this);
1275     this._undoManager = new Type.
1276         UndoManager(this);
1277     //this._content = new Type.
1278         Content(this);
1279     this._caret = new Type.
1280         Caret(this);
1281     this._selection = new Type.
1282         Selection(this);
1283     this._input = new Type.
1284         Input(this);
1285
1286     // Trigger events
1287     Type.trigger('ready', this);
1288
1289     (function () {
1290
1291         /**
1292          * Allows fast detection if an
1293          * object is a Type Editor
1294          * instance (or class)
1295          * @type {boolean}
1296          */
1297         this.typeEditor = true;
1298
1299         /**
1300          * Holds the default options for
1301          * every editor. These options
1302          * will be extended by the options
1303          * passed to each instance
1304          * on instantiation.
1305          */
1306     })
1307
1308     // Set settings for this editor
1309     this._root = null;
1310     this.options(options);
1311
1312     // Set up core modules
1313     this._writer = new Type.
1314         Writer(this);
1315     this._formatter = new Type.
1316         Formatter(this);
1317     this._undoManager = new Type.
1318         UndoManager(this);
1319     //this._content = new Type.
1320         Content(this);
1321     this._caret = new Type.
1322         Caret(this);
1323     this._selection = new Type.
1324         Selection(this);
1325     this._input = new Type.
1326         Input(this);
1327
1328     // Trigger events
1329     Type.trigger('ready', this);
1330
1331     (function () {
1332
1333         /**
1334          * Allows fast detection if an
1335          * object is a Type Editor
1336          * instance (or class)
1337          * @type {boolean}
1338          */
1339         this.typeEditor = true;
1340
1341         /**
1342          * Holds the default options for
13
```

```

95  *      this value will be set to
96  *      the key of the given first
97  *      parameter. Any arbitrary
98  *      value can be set.
99  *      @returns {Type|*} Returns the
100  *      type instance if you set an
101  *      option or the according
102  *      value if you get an option
103  */
104  this.options = function (options,
105  value) {
106
107  // Load default options if there
108  // are no instance options
109  yet
110  this._options = this._options ||
111  Type.Utilities.extend({},
112  this._defaultOptions);
113
114  // Pass a single option name to
115  // fetch it
116  if (typeof options === "string"
117  && arguments.length === 1)
118  {
119  return this._options[options];
120  }
121
122  // Pass an option name and a
123  // value to set it
124  if (typeof options === "string"
125  && arguments.length === 2)
126  {
127  options = {options: value};
128  }
129
130  // Pass an object of key-values
131  // to set them
132  if (typeof options === "object")
133  {
134  Type.Utilities.extend(this._
135  options, options);
136  }
137
138  // If the el option has been
139  // passed copy it for quick
140
141  *      access
142  *      if (options.el) {
143  *      this._root = options.el;
144  *      }
145  *      // Chaining
146  *      return arguments.length ? this :
147  *      this._options;
148  *      //return this;
149  *      };
150  /**
151  *      Creates a {Type.DomWalker} that
152  *      ist constrained to this
153  *      instance's root element unless
154  *      you explicitly pass a
155  *      constrainingNode as argument.
156  *      All other DomWalker options
157  *      can also be passed to this as
158  *      usual.
159  *      @param {Node} node - Any DOM {
160  *      Node} to be set as starting
161  *      node for the DomWalker
162  *      @param {Node|string|Function|{
163  *      constrainingNode: Node,
164  *      filter: string|Function}} [
165  *      options]
166  *      See {Type.DomWalker} for a
167  *      description of possible
168  *      arguments
169  *      @returns {Type.DomWalker}
170  */
171  this.createDomWalker = function (
172  node, options) {
173  options = Type.DomWalker.
174  loadOptions(options || {});
175  options.constrainingNode =
176  options.constrainingNode ||
177  this._root;
178  return new Type.DomWalker(node,
179  options);
180  };
181
182  *      Getter for this instance's
183  *      element, i.e. the
184  *      element that contains this
185  *      editor's text.
186  *      @returns {Element}
187  */
188  this.getRoot = function () {
189  return this._root;
190  };
191
192  *      Getter for this instance's
193  *      caret. {Type.Caret}
194  */
195  this.getCaret = function () {
196  return this._caret;
197  };
198
199  *      Getter for this instance's
200  *      selection.
201  *      @returns {Type.Selection}
202  */
203  this.getSelection = function () {
204  return this._selection;
205  };
206
207  *      Getter for this instance's
208  *      content.
209  *      @returns {Type.UndoManager}
210  */
211  this.getUndoManager = function ()
212  {
213  return this._undoManager;
214  };
215
216  *      Getter for this instance's
217  *      content.
218  *      @returns {Type.Content}
219  */

```

```

186 //this.getContent = function () { 204
187 // return this._content; 205
188 //}; 206
189 207
190 /** 208
191 * Getter for this instance's 209
192 * @returns {Type.Writer} 210
193 */ 211
194 this.getWriter = function () { 212
195 return this._writer; 213
196 }; 214
197 215
198 /** 216
199 * Getter for this instance's 217
200 * @returns {Formatter} 218
201 */ 219
202 this.getFormatter = function () { 220
203 return this._formatter;

```

```

    };
    /**
     * Getter for this instance's
     * input.
     * @returns {Type.Input}
     */
    this.setInput = function () {
        return this._input;
    };
    }.call(Type.prototype);
    /**
     * Exposes Type's prototype as
     * jQuery-style shorthand
     * variable
     * @type {Object}
     */
    Type.fn = Type.prototype;

```

Listing 1: core.js

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6   * Holds messages for developing and
7   * debugging Type
8   * @constructor
9   Type.Development = function () {
10 };
11
12 (function () {
13
14 /**
15  * Prints a message to the console
16  * if the browser's
17  * console offers the log method.

```

```

32  * @param {...*} messages — Any
33  *   number and type of arguments
34  *   you want to pass to console
35  *   .debug
36  Type.Development.debug = function
37  (messages) {
38     if (console && console.debug) {
39         console.debug.apply(console,
40             arguments)
41     }
42     return Type.Development;
43 };
44
45 }).call(Type.Development);

```

```

module.exports = Type.Development;

```

Listing 2: development.js

```

1 'use strict';
2
3 var Type = require('./core');
4
5
6
7
8
9 module.exports = Type.Settings;
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Listing 3: settings.js

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Listing 4: oop.js

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

23	this.eventCallbacks[eventName] =	50	* @param {...*} params -	81	*/	81	this.on = function (eventName, cb)
24	this.eventCallbacks[Arbitrary parameters you	82	{		
25	eventName] [];	51	wish to pass to the		this.eventCallbacks = this.		
26	push(cb);	52	callbacks	83	eventCallbacks {};		
27	return this;	53	* @returns {Type}	84	this.eventCallbacks[eventName] =		
28	};	54	*/		this.eventCallbacks[
29	/**		this.trigger = function (eventName	85	eventName] [];		
30	var i;	55	, params) {		push(cb);		
31	this.eventCallbacks = this.	56	eventCallbacks {};	86	return this;		
32	if (this.eventCallbacks[57	eventName]) {	87	};		
33	for (i = 0; i < this.	58	eventCallbacks[eventName	88	/**		
34].length; i += 1) {		this.eventCallbacks[89	* Unregister a callback for a		
35	eventName][i].apply(59	this, params);	90	global Type event		
36	}	60	}	91	* @param {String} eventName - The		
37	return this;	61	}	92	name of the event on which		
38	};	62	};	93	you wish the		
39	this.off = function (eventName, cb	63		94	for which you no longer		
40) {	64	this.eventCallbacks = this.		wish to call the function		
41	var index = this.eventCallbacks[65	eventName] ? this.	95	* @param {Function} cb - The		
42	eventCallbacks[eventName].	66	indexOf(cb) : -1;		function you no longer wish		
43	if (index > -1) {	67	}	96	to be called		
44	this.eventCallbacks[eventName	68	}.call((Type.fn);	97	@returns {Type}		
45].splice(index, 1);	69	/**		*/		
46	return this;	70	* Global Type.js events		this.off = function (eventName, cb		
47	};	71	* Todo Remove code duplication) {		
48	/**	72	(function () {	98	this.eventCallbacks = this.		
49	Trigger a Type specific event	73	}	99	eventCallbacks {};		
	to call all callbacks for	74	return this;		var index = this.eventCallbacks[
		75			eventName] ? this.		
		76			eventCallbacks[eventName].		
		77			indexOf(cb) : -1;		
		78			if (index > -1) {		
		79			this.eventCallbacks[eventName		
		80].splice(index, 1);		
					return this;		
					};		
					/**		
					* Trigger a Type specific event		
					to call all callbacks for		
					* @param {String} eventName - The		
					name of the event on which		
					you wish to		
					call its callbacks for		
					* Trigger a global Type event to		
					call all callbacks for		

```

108 * @param {String} eventName - The
109 *   name of the event on which
110 *   you wish to
111 *   call its callbacks for
112 * @param {...*} params -
113 *   Arbitrary parameters you
114 *   wish to pass to the
115 *   callbacks
116 * @returns {Type}
117 */
118
119 this.trigger = function (eventName
120 , params) {
121   var i;
122   this.eventCallbacks = this.
123   eventCallbacks || {};
124   if (this.eventCallbacks[
125   eventName]) {
126     for (i = 0; i < this.
127     eventCallbacks[eventName
128     ].length; i += 1) {
129       this.eventCallbacks[
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

Listing 5: event_api.js


```

67      * Call a method from an object (
68      usually a plugin). If the
69      method returns the plugin,
70      return this type instance
71      instead.
72      * If the given method name is not
73      a method in the object,
74      call the
75      given callback.
76      * callMethodFrom purpose is to
77      provide a shorthand way to
78      expose
79      the API of a plugin as API of
80      Type
81      * @param module
82      * @param method
83      * @param params
84      * @param fallback
85      * @returns {Type|*}
86      */
87      this.callMethodFrom = function (
88      module, method, params,
89      fallback) {
90      var result = null;
91      if (module.hasOwnProperty(method)) {
92      result = module[method].apply(
93      module, params);
94      } else if (fallback) {
95      result = fallback.apply(module
96      , [method].concat(params));
97      } else {
98      throw new Error('Method ' +
99      method + 'cannot be found
100      in given module');
101      }
102      return result === module ? this
103      : result;
104      };
105      }).call(Type.fn);

```

Listing 6: plugin_api.js

```

1  'use strict';
2
3  var Type = require('./core');
4
5  Type.Environment = function () {
6  };
7
8  (function () {
9
10     /**
11     * Is the user's computer a
12     * Macintosh computer
13     * @type {boolean}
14     */
15     Type.Environment.mac = navigator.
16     appVersion.indexOf("Mac") !==
17     -1;
18     module.exports = Type.Environment;

```

Listing 7: environment.js

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6  * @constructor
7  */
8  Type.Utilities = function () {
9  };
10
11  (function () {
12
13     /**
14     * This behaves similar to jQuery's
15     * extend method. Writes all
16     * from the objects passed as
17     * copyFrom to the object
18     * passed as copyTo.
19     * Copying starts from left to
20     * right and will overwrite
21     * each setting
22     * subsequently.
23     * @param {Object} copyTo
24     * @param {...Object} copyFrom
25     * @returns {Object}
26     */
27     Type.Utilities.extend = function (
28     copyTo, copyFrom) {
29     var i, key;
30     for (i = 1; i < arguments.length;
31     i++) {
32     for (key in arguments[i]) {
33     if (arguments[i][key] ===
34     arguments[0][key]) {
35     }
36     }
37     }
38     }

```

```

31     }
32     return arguments[0];
33 };
34
35 /**
36  * Tests and returns if a given
37  * object is a function
38  * instance
39  * todo this should be called
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

```

isFunctionInstance otherwise
typeof obj === 'Function'
should be used

```

38
39 * @param obj
40 * @returns {boolean}
41
42 Type.Utilities.isFunction =
  function(obj) {
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

```

return !(obj && obj.constructor
&& obj.call && obj.apply);
};
}).call(Type.Utilities);
module.exports = Type.Utilities;
Listing 8: utilities.js

```

50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

```

el.parentNode.removeChild(el);
return this;
};
/**
 * Will remove a node and each
 * parent (recursively) if
 * removing
 * leaves the parent with no *
 * visible* content
 *
 * @param {Node} node - The node
 * to remove
 * @param {Node} [constrainingNode
 *] - The algorithm will stop
 * and
 * not remove this node if it
 * reaches it
 * @returns {Node|null} - Will
 * return the parent node where
 * this
 * algorithm stopped (The node
 * it did *not* delete)
 */
Type.DomUtilities.removeVisible =
function (node,
constrainingNode) {
var parent = node.parentNode;
if (node === constrainingNode)
return node;
if (node === document.body)
return node;
if (parent === null) return null
;

```

29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

Type.DomUtilities.singleTag =
/^(?!(\\w-|+|s*`/?>(?:<\\|>|
\$);
/**
 * Todo Use me wherever you find
 * document.createElement or
 * this.elementsContainer
 * @param {string} tagName
 * @param {string} [className]
 * @returns {Element}
 */
Type.DomUtilities.addElement =
function (tagName, className)
{
var el = document.createElement(
tagName);
if (className) el.className =
Type.Settings.prefix +
className;
this.getElementsContainer().
appendChild(el);
return el;
};
/**
 * Removes a DOM element
 * @param {Element} el
 * @returns {*}
 */
Type.DomUtilities.removeElement =
function (el) {

```

29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49

```

'use strict';
var Type = require('./core');
/**
 * @constructor
 */
Type.DomUtilities = function () {
};
(function () {
/**
 * The id attribute of the
 * container element where all
 * elements including carets and
 * input fields of type will be
 * appended to
 * @type {string}
 * @private
 */
Type.DomUtilities._containerId =
Type.Settings.prefix +
'container';
/**
 * Matches a single HTML tag
 * @type {RegExp}
 * @private
 */

```

69 parent.removeChild(node);
70 if (!this.isVisible(parent))
71     return this.removeVisible(
72         parent, constrainingNode)
73     ;
74 return parent;
75 };
76 /**
77  * Recursively unwraps the given
78  * tag from the element passed
79  * an all its children
80  * Note to self and future
81  * developers, querySelectorAll
82  * can be used for this when
83  * we drop IE 8 support.
84  *
85  * @param el
86  * @param tag
87  * @param deep
88  * @returns {Type.DomUtilities}
89
90 Type.DomUtilities.removeTag =
91 function (el, tag, deep) {
92     var i;
93     if (deep && el.childNodes.length
94         ) {
95         for (i = 0; i < el.childNodes.
96             length; i += 1) {
97             this.removeTag(el.childNodes
98                 [i], tag, deep);
99         }
100     }
101     if (el.nodeType === 1 && el.
102         tagName.toLowerCase() ===
103         tag.toLowerCase()) {
104         this.unwrap(el);
105     }
106     return this;
107 };
108 /**
109  * Converts a string of HTML to a
110  * corresponding {NodeList}
111
112 parent.removeChild(node);
113 if (!this.isVisible(parent))
114     return this.removeVisible(
115         parent, constrainingNode)
116     ;
117 return parent;
118 };
119 /**
120  * Recursively unwraps the given
121  * tag from the element passed
122  * an all its children
123  * Note to self and future
124  * developers, querySelectorAll
125  * can be used for this when
126  * we drop IE 8 support.
127  *
128  * @param el
129  * @param tag
130  * @param deep
131  * @returns {Type.DomUtilities}
132
133 Type.DomUtilities.parseHTML =
134 function(htmlString) {
135     var fragment = document.
136         createDocumentFragment(),
137         div = fragment.appendChild(
138             document.createElement(
139                 'div'));
140     div.innerHTML = htmlString;
141     return div.childNodes;
142 };
143 /**
144  * By Dave Atchley, taken (and
145  * modified) from
146  * {@link https://gist.github.com/
147     datchley/11383482}
148  * No license given, I asked for
149  * the license by mail.
150  * Still waiting.
151  *
152  * @param tag
153  * @param elms
154  * @returns {Element}
155
156 Type.DomUtilities.wrap = function
157 (tag, elms) {
158     // Even out parameters
159     elms = elms.length ? elms : [
160         elms];
161     // Prepare vars and cache the
162     // current parent
163     // and sibling of the first
164     // element.
165     var el = elms[0],
166         parent = el.parentNode,
167         sibling = el.nextSibling,
168         wrapper = document.
169             createElement(tag),
170         i;
171     // If the first element had a
172     // sibling, insert the wrapper
173     // sibling to maintain the HTML
174     // structure; otherwise, just
175     // append it
176     // to the parent.
177     if (sibling) {
178         parent.insertBefore(wrapper,
179             sibling);
180     }
181     else {
182         parent.appendChild(wrapper);
183     }
184     // Move all elements to the
185     // wrapper. Each element is
186     // automatically removed from
187     // its current parent and
188     // from the elms array.
189     for (i = 0; i < elms.length; i
190         += 1) {
191         wrapper.appendChild(elms[i]);
192     }
193     // Remove the tag we want to
194     // wrap from TypeContents
195     // so we don't have the same tag
196     // nested
197     for (i = 0; i < elms.length; i
198         += 1) {
199         this.removeTag(elms[i], tag,
200             true);
201     }
202     // Return newly created element
203     // return wrapper;
204 };
205 /**
206  * Todo use this.moveAfter()
207  * @param {Node} el

```

```

164 * @returns {Type.DomUtilities}
165 */
166 Type.DomUtilities.unwrap =
167   function (el) {
168     var next = el.nextSibling,
169     parent = el.parentNode,
170     childNodes = el.childNodes;
171
172     if (next) {
173       while (childNodes.length) {
174         parent.insertBefore(
175           lastChild, next);
176       }
177     } else {
178       while (childNodes.length) {
179         parent.appendChild(
180           firstChild);
181       }
182     }
183     parent.removeChild(el);
184     parent.normalize();
185     return this;
186   };
187 /**
188 *
189 * @param reference
190 * @param elems
191 * @returns {*}
192 */
193 Type.DomUtilities.moveAfter =
194   function (reference, elems) {
195     var i;
196     var next = reference.nextSibling,
197     parent = reference.parentNode;
198     elems = !elems.length ? [elems]
199       : Array.prototype.slice.
200
201
202     call(elems, 0);
203
204     if (next) {
205       for (i = 0; i < elems.length;
206         i += 1) {
207         parent.insertBefore(elems[i],
208           next);
209       }
210     } else {
211       for (i = 0; i < elems.length;
212         i += 1) {
213         parent.appendChild(elems[i])
214       }
215     }
216     return this;
217   };
218 /**
219 * Todo move to dom walker??
220 *
221 * @param {Node} el
222 * @param {String} selector
223 * @param {Node} [constrainingNode]
224 * @returns {HTMLElement|null}
225 */
226 Type.DomUtilities.parent =
227   function (el, selector,
228     constrainingNode) {
229     while (el.parentNode && (!
230       constrainingNode || el !==
231       constrainingNode)) {
232       if (this.matches(el, selector)
233         ) {
234         return el;
235       }
236       el = el.parentNode;
237     }
238     return null;
239   };
240 /**
241 *
242 * @returns {Element}
243 */
244 Type.DomUtilities.
245   getElementsContainer =
246   function () {
247
248     Returns true if el matches the
249     CSS selector given as second
250     argument,
251     otherwise false
252
253     * Todo http://davidwalsh.name/
254     element-matches-selector
255
256     * @param el
257     * @returns {boolean}
258 */
259 Type.DomUtilities.matches =
260   function (el, selector) {
261     var _matches = (el.matches || el.
262       .matchesSelector || el.
263       msMatchesSelector || el.
264       mozMatchesSelector || el.
265       webkitMatchesSelector || el.
266       oMatchesSelector);
267
268     if (!_matches) {
269       return _matches.call(el,
270         selector);
271     } else {
272       var nodes = el.parentNode.
273         querySelectorAll(selector
274         );
275       for (var i = nodes.length; i
276         --;) {
277         if (nodes[i] === el)
278           return true;
279       }
280       return false;
281     }
282   };
283 /**
284 *
285 * @returns {Element}
286 */
287 Type.DomUtilities.
288   getElementsContainer =
289   function () {

```

```

264 var container = window.document.
265   getElementById(this.
266   _containerId);
267 if (container === null) {
268   container = window.document.
269   createElement('div');
270   container.setAttribute('id',
271   this._containerId);
272   window.document.body.
273   appendChild(container);
274   return container;
275 }
276 /**
277 * @param {Node} container
278 * @param {Node} node
279 * @returns {boolean}
280 */
281 Type.DomUtilities.containsButIsnt
282 = function (container, node)
283 {
284   return container !== node &&
285   container.contains(node);
286 }
287 /**
288 * @param {Node} node - The node to
289   be used as the starting point
290   for the
291   first traversal operation.
292 * @param {Object|Node} [options] -
293   If an object is passed, it
294   should
295   contain settings determining
296   what node to return, see
297   specifics
298   below. If a {Node} is passed,
299   this acts as options.
300   constrainingNode
301   @param {Function|string} [options
302   .filter] - nextNode traverses
303   the DOM tree and passes each
304   node to this function. This
305   function
306   should return true if the
307   node passed is a node that we
308   look for
309   or false otherwise. E.g. if
310   we want to find the next text
311   node
312   in the tree, the function
313   should check if the node
314   passed is of
315   nodeType === 3. If this
316   parameter is not set, any node
317   found
318   will be returned.
319   todo allow css selectors to
320   be used for traversal
321   @param {Node} [options.
322   constrainingNode] While
323   traversing the DOM,
324   other node
325   * @param {Node} b - A DOM node to
326   compare with the given
327   other node
328   * @returns {number} - Returns -1
329   if a precedes b, 1 if it is
330   the
331   other way around and 0 if
332   they are equal.
333 */
334 Type.DomUtilities.order = function
335 (a, b) {
336   if (a === b) {
337     return 0;
338   }
339   if (a.compareDocumentPosition(b)
340   & Node.
341   DOCUMENT_POSITION_FOLLOWING
342   ) {
343     return -1;
344   }
345   return 1;
346 }
347 /**
348 * @param {Node} a - A DOM node to
349   compare with the given
350   other node
351   * @param {Node} b - A DOM node to
352   compare with the given
353   other node
354   * @returns {boolean}
355 */
356 Type.DomUtilities.isVisible =
357 function (el) {
358   return !!(el.offsetHeight);
359 }
360 /**
361 * Compares the document positions
362   of two DOM nodes
363   @param {Node} a - A DOM node to
364   compare with the given
365   other node
366   * @param {Node} b - A DOM node to
367   compare with the given
368   other node
369   * @returns {boolean}
370 */
371 Type.DomUtilities.isVisible =
372 function (el) {
373   return !!(el.offsetHeight);
374 }
375 /**
376 * Compares the document positions
377   of two DOM nodes
378   @param {Node} a - A DOM node to
379   compare with the given
380   other node
381   * @param {Node} b - A DOM node to
382   compare with the given
383   other node
384   * @returns {boolean}
385 */
386 Type.DomUtilities.isVisible =
387 function (el) {
388   return !!(el.offsetHeight);
389 }
390 /**
391 * Compares the document positions
392   of two DOM nodes
393   @param {Node} a - A DOM node to
394   compare with the given
395   other node
396   * @param {Node} b - A DOM node to
397   compare with the given
398   other node
399   * @returns {boolean}
400 */
401 Type.DomUtilities.isVisible =
402 function (el) {
403   return !!(el.offsetHeight);
404 }
405 /**
406 * Compares the document positions
407   of two DOM nodes
408   @param {Node} a - A DOM node to
409   compare with the given
410   other node
411   * @param {Node} b - A DOM node to
412   compare with the given
413   other node
414   * @returns {boolean}
415 */
416 Type.DomUtilities.isVisible =
417 function (el) {
418   return !!(el.offsetHeight);
419 }
420 /**
421 * Compares the document positions
422   of two DOM nodes
423   @param {Node} a - A DOM node to
424   compare with the given
425   other node
426   * @param {Node} b - A DOM node to
427   compare with the given
428   other node
429   * @returns {boolean}
430 */
431 Type.DomUtilities.isVisible =
432 function (el) {
433   return !!(el.offsetHeight);
434 }
435 /**
436 * Compares the document positions
437   of two DOM nodes
438   @param {Node} a - A DOM node to
439   compare with the given
440   other node
441   * @param {Node} b - A DOM node to
442   compare with the given
443   other node
444   * @returns {boolean}
445 */
446 Type.DomUtilities.isVisible =
447 function (el) {
448   return !!(el.offsetHeight);
449 }
450 /**
451 * Compares the document positions
452   of two DOM nodes
453   @param {Node} a - A DOM node to
454   compare with the given
455   other node
456   * @param {Node} b - A DOM node to
457   compare with the given
458   other node
459   * @returns {boolean}
460 */
461 Type.DomUtilities.isVisible =
462 function (el) {
463   return !!(el.offsetHeight);
464 }
465 /**
466 * Compares the document positions
467   of two DOM nodes
468   @param {Node} a - A DOM node to
469   compare with the given
470   other node
471   * @param {Node} b - A DOM node to
472   compare with the given
473   other node
474   * @returns {boolean}
475 */
476 Type.DomUtilities.isVisible =
477 function (el) {
478   return !!(el.offsetHeight);
479 }
480 /**
481 * Compares the document positions
482   of two DOM nodes
483   @param {Node} a - A DOM node to
484   compare with the given
485   other node
486   * @param {Node} b - A DOM node to
487   compare with the given
488   other node
489   * @returns {boolean}
490 */
491 Type.DomUtilities.isVisible =
492 function (el) {
493   return !!(el.offsetHeight);
494 }
495 /**
496 * Compares the document positions
497   of two DOM nodes
498   @param {Node} a - A DOM node to
499   compare with the given
500   other node
501   * @param {Node} b - A DOM node to
502   compare with the given
503   other node
504   * @returns {boolean}
505 */
506 Type.DomUtilities.isVisible =
507 function (el) {
508   return !!(el.offsetHeight);
509 }
509

```

Listing 9: dom_utilities.js

```

20 *      this method will check nodes'
21 *      parents and parents' parents.
22 *      By
23 *      passing a DOM node as this
24 *      parameter, traversing up will
25 *      stop at
26 *      this node and return null.
27 *      This is useful when you want
28 *      to permit
29 *      traversing outside the editor
30 *      's root node.
31 * @constructor
32 Type.DomWalker = function (node,
33 options) {
34   this.setNode(node);
35   this.options(options);
36 };
37 (function () {
38 /**
39 * Returns the next node in the
40 * document flow and sets the
41 * internal reference
42 * to the current node to that
43 * node.
44 * @returns {null|Node}
45 */
46 this.next = function (returnMe) {
47   return this._setNodeIfNotNull(
48     Type.DomWalker._nextNode(
49       this._node, this._options,
50       returnMe));
51 };
52 /**
53 * Returns the next node in the
54 * document flow but does not
55 * set the internal
56 * reference to the current node
57 * to that node.
58 * @returns {null|Node}
59 */
60 this.fetchNext = function (returnMe) {
61   return this._setNodeIfNotNull(
62     Type.DomWalker._fetchNextNode(
63       this._node, this._options,
64       returnMe));
65 };
66 /**
67 * Returns the first child node
68 * matching the given filter or
69 * the node passed itself
70 * if it matches the filter too.
71 * Sets the internal reference
72 * for the current node to
73 * the node found.
74 * @returns {null|Node}
75 */
76 this.firstChild = function (filter) {
77   return this._setNodeIfNotNull(
78     Type.DomWalker._firstChildNode(
79       this._node, this._options,
80       filter));
81 };
82 /**
83 * Returns the last child node
84 * matching the given filter or
85 * the node passed itself
86 * if it matches the filter too.
87 * Sets the internal reference
88 * for the current node to
89 * the node found.
90 * @returns {null|Node}
91 */
92 this.last = function () {
93   return this._setNodeIfNotNull(
94     Type.DomWalker._lastNode(
95       this._node, this._options,
96       filter));
97 };
98 /**
99 * Sets the internal node from
100 * which traversal is made to
101 * the given node.
102 * @param {Node} node
103 */
104 this.setNode = function (node) {
105   if (!node.nodeType) {
106     throw new Error('The given
107       node is not a DOM node');
108   }
109   this._node = node;
110   return this;
111 };
112 /**
113 * Sets the options used for
114 * traversal by this walker
115 * @param options

```

```

106 * @returns {*}
107 */
108 this.options = function (options)
109 {
110     this._options = Type.DomWalker.
111     loadOptions(options);
112     return this;
113 };
114 /**
115 * Returns the current node the
116 * walker is on.
117 * @returns {Node}
118 */
119 this.getNode = function () {
120     return this._node;
121 };
122 /**
123 * Will set this _node to the
124 * given node unless null is
125 * passed.
126 * Will also return either null or
127 * the node, depending on what
128 * has been passed. This method is
129 * used to process the return
130 * values by the DomWalker
131 * traversal methods.
132 * @param {Node|null} node
133 * @returns {Node|null}
134 * @private
135 */
136 this._setNodeIfNotNull = function
137 (node) {
138     if (node === null) {
139         return null;
140     }
141     this._node = node;
142     return node;
143 };
144 /**
145 * If no options parameter has
146 * been passed
147 * options = options || {};
148 */
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211

```

212	// If a node has been passed as	243	} return filter;	264	* @param {Node} [options,
213	options parameter	244	};	265	constrainingNode] While
214	if (options.nodeType) {	245	options = {constrainingNode:	266	traversing the DOM,
215	options};	246	options};	267	this method will check
216	}	247	options};	268	nodes' parents and parents',
217	// If a function has been passed	248	options};	269	parents. By
218	as options parameter	249	options};	270	passing a DOM node as this
219	if (typeof options === 'string',	250	options};	271	parameter, traversing up
220	Type.Utilities.	251	options = {filter: options};	272	will stop at
221	isFunction(options)) {	252	options = {filter: options};	273	this node and return null.
222	options = {filter: options};	253	options = {filter: options};	274	This is useful when you want
223	// Load internal filter function	254	options = {filter: options};	275	to permit
224	if filter param is a	255	options = {filter: options};	276	traversing outside the
225	string	256	options = {filter: options};	277	editor's root node.
226	if (options.filter) {	257	options = {filter: options};	278	@param {boolean} [returnMe]
227	options.filter = Type.	258	options = {filter: options};	279	This should not be passed by
228	DomWalker._loadFilter(259	options = {filter: options};	280	the
229	DomWalker._loadFilter(260	options = {filter: options};	281	programmer, it is used
230	options.filter);	261	options = {filter: options};	282	internally for recursive
231	}	262	options = {filter: options};	283	function calls to
232	// Return processed options	263	options = {filter: options};	284	determine if the current
233	return options;	264	options = {filter: options};	285	node should be returned or
234	};	265	options = {filter: options};	286	not. If the
235	/**	266	options = {filter: options};	287	programmer passes a node
236	* @param filter	267	options = {filter: options};	288	and does *not* pass this
237	* @returns {*}	268	options = {filter: options};	289	argument, the
238	* @private	269	options = {filter: options};	290	node passed will not be
239	Type.DomWalker._loadFilter =	270	options = {filter: options};	291	considered for returning.
240	function (filter) {	271	options = {filter: options};	292	After that,
241	var funcName;	272	options = {filter: options};	293	internally, this will be
242	if (typeof filter === 'string')	273	options = {filter: options};	294	set to true and be passed on
243	{	274	options = {filter: options};	295	with the
244	funcName = Type.DomWalker.	275	options = {filter: options};	296	next node in the DOM to a
245	_filterFunctions[filter];	276	options = {filter: options};	297	recursive call. The node
246	return Type.DomWalker[funcName]	277	options = {filter: options};	298	then passed to
247	};	278	options = {filter: options};	299	this method might be the
248		279	options = {filter: options};	300	node we are looking for, so
249		280	options = {filter: options};	301	having this
250		281	options = {filter: options};	302	set to true will return
251		282	options = {filter: options};	303	that node (given that the
252		283	options = {filter: options};	304	filter
253		284	options = {filter: options};	305	also returns true for that
254		285	options = {filter: options};	306	node)
255		286	options = {filter: options};	307	@returns {null Node} The next
256		287	options = {filter: options};	308	node in the DOM tree found
257		288	options = {filter: options};	309	or null
258		289	options = {filter: options};	310	if none is found for the
259		290	options = {filter: options};	311	option.filter criteria or
260		291	options = {filter: options};	312	
261		292	options = {filter: options};	313	
262		293	options = {filter: options};	314	
263		294	options = {filter: options};	315	
264		295	options = {filter: options};	316	
265		296	options = {filter: options};	317	
266		297	options = {filter: options};	318	
267		298	options = {filter: options};	319	
268		299	options = {filter: options};	320	
269		300	options = {filter: options};	321	
270		301	options = {filter: options};	322	
271		302	options = {filter: options};	323	
272		303	options = {filter: options};	324	
273		304	options = {filter: options};	325	
274		305	options = {filter: options};	326	
275		306	options = {filter: options};	327	
276		307	options = {filter: options};	328	
277		308	options = {filter: options};	329	
278		309	options = {filter: options};	330	
279		310	options = {filter: options};	331	
280		311	options = {filter: options};	332	
281		312	options = {filter: options};	333	
282		313	options = {filter: options};	334	
283		314	options = {filter: options};	335	
284		315	options = {filter: options};	336	
285		316	options = {filter: options};	33	


```

281 *      options.constrainingNode
282 *      has been hit.
283 */
284 Type.DomWalker._nextNode =
285   function (node, options,
286     returnMe) {
287
288     // For later use
289     var parent = node.parentNode;
290
291     // If a node is found in this
292     call, return it, stop the
293     recursion
294     if (returnMe === true && (!
295       options.filter || options.
296       filter(node))) {
297       return node;
298     }
299
300     // 1. If this node has children,
301     go down the tree
302     if (node.childNodes.length) {
303       return Type.DomWalker.
304         _nextNode(node.childNodes
305           [0], options, true);
306     }
307
308     // 2. If this node has siblings,
309     move right in the tree
310     if (node.nextSibling !== null) {
311       return Type.DomWalker.
312         _nextNode(node.
313           nextSibling, options,
314           true);
315     }
316
317     // 3. Move up in the node's
318     parents until a parent has
319     a sibling or the
320     constrainingNode is hit
321     while (parent !== options.
322       constrainingNode) {
323       if (parent.nextSibling !==
324         null) {
325         return Type.DomWalker.
326           _nextNode(parent.
327             nextSibling, options,
328             true);
329       }
330     }
331
332     // We have not found a node we
333     were looking for
334     return null;
335   };
336
337 /** Traverses the DOM tree and
338     finds the previous node
339     before the node passed
340     as first argument. Will
341     traverse the children,
342     siblings and parents,
343     siblings (in that order) to
344     find the next node in the
345     DOM tree as
346     displayed by the document flow.
347
348     * @param {Node} node - The node
349     from which the search should
350     start
351     * @param {Object|Node} [options]
352     - If an object is passed, it
353     should
354     * contain settings
355     determining what node to
356     return, see specifics
357     below. If a {Node} is
358     passed, this acts as options.
359     .constrainingNode
360     * @param {Function} [options.
361     filter] - nextNode traverses
362     the
363     DOM tree and passes each
364     node to this function. This
365     function
366     should return true if the
367     node passed is a node that
368     we look for
369     * or false otherwise. E.g. if

```

```

we want to find the next
text node
* in the tree, the function
should check if the node
passed is of
* nodeType === 3. If this
parameter is not set, any
node found
* will be returned.
* @param {Node} [options.
constrainingNode] While
traversing the DOM,
this method will check
nodes' parents and parents',
parents. By
* passing a DOM node as this
parameter, traversing up
will stop at
* this node and return null.
This is useful when you want
to permit
* traversing outside the
editor's root node.
* @param {boolean} [returnMe]
This should not be passed by
the
* programmer, it is used
internally for recursive
function calls to
* determine if the current
node should be returned or
not. If the
* programmer passes a node
and does *not* pass this
argument, the
node passed will not be
considered for returning.
After that,
* internally, this will be
set to true and be passed on
with the
* next node in the DOM to a
recursive call. The node
then passed to
this method might be the
node we are looking for, so
having this
* set to true will return

```

```

347         that node (given that the
371         filter
372         also returns true for that
348         node)
373         * @returns {null|Node} The next
374         node in the DOM tree found
375         or null
376         * if none is found for the
377         options.filter criteria or
378         options.constrainingNode
379         has been hit.
380         */
381         Type.DomWalker._prevNode =
382         function (node, options,
383         returnMe) {
384             // For later use
385             var parent = node.parentNode;
386             // If a node is found in this
387             call, return it, stop the
388             recursion
389             if (returnMe === true && (!
390             options.filter || options.
391             filter(node))) {
392                 return node;
393             }
394             // 1. If this node has children,
395             go down the tree
396             if (node.childNodes.length) {
397                 return Type.DomWalker.
398                 _prevNode(node.lastChild,
399                 options, true);
400             }
401             // 3. Move up in the node's
402             parents until a parent has
403             a sibling or the
404             constrainingNode is hit
405             while (parent !== options.
406             constrainingNode) {
407                 if (parent.previousSibling !==
408                 null) {
409                     return Type.DomWalker.
410                     _prevNode(parent.
411                     previousSibling,
412                     options, true);
413                 }
414                 parent = parent.parentNode;
415             }
416             // We have not found a node we
417             were looking for
418             return null;
419         };
420         /**
421         * Returns true if a given node is
422         a text node
423         * @param {Node} node The node to
424         be checked.
425         * @returns {boolean}
426         * @private
427         */
428         Type.DomWalker._isTextNode =
429         function (node) {
430             return node.nodeType === Node.
431             TEXT_NODE;
432         };
433         /**
434         * Returns true if a given node is
435         a text node and its
436         contents are not
437         entirely whitespace.
438         * @param {Node} node The node to
439         be checked.
440         * @returns {boolean}
441         * @private
442         */
443         Type.DomWalker._isVisible =
444         function (node) {
445             return !node.offsetHeight;
446         };
447         /**
448         * Returns true if the given node
449         is visible to the user.
450         * @param {Element} node - The
451         node to be checked
452         * @returns {boolean}
453         * @private
454         */
455         Type.DomWalker._isVisible =
456         function (node) {
457             return !node.offsetHeight;
458         };
459         /**
460         * Returns true if a given node is
461         displayed as text on the
462         screen
463         * @param {Node} node The node to
464         be checked.
465         * @returns {boolean}
466         * @private
467         */
468         Type.DomWalker._resemblesText =
469         function (node) {
470             return node.nodeName.
471             toLowerCase() === 'br'
472             , || Type.DomWalker.
473             _isTextNodeWithContents(
474             node);
475         };
476         /**
477         * Returns true if the given node
478         is visible to the user.
479         * @param {Element} node - The
480         node to be checked
481         * @returns {boolean}
482         * @private
483         */
484         Type.DomWalker._isVisible =
485         function (node) {
486             return !node.offsetHeight;
487         };

```

```

430 }) . call (Type . DomWalker) ;
431
432
433 module . exports = Type . DomWalker ;
Listing 10: dom_walker.js
1  'use strict' ;
2  var Type = require ( './core' ) ;
3
4  Type . TextWalker = function () {
5  } ;
6
7  (function () {
8
9  /**
10 * @param fromNode
11 * @param toNode
12 * @param fromOffset
13 * @param toOffset
14 * @returns {*}
15 */
16 Type . TextWalker . offset = function
17 (fromNode, toNode, fromOffset
18 , toOffset) {
19
20 var dom = new Type . DomWalker (
21 fromNode, 'textual' ) ,
22 node = dom . next (true) ,
23 offsetWalked = 0 ;
24
25 fromOffset = fromOffset || 0 ;
26 toOffset = toOffset || 0 ;
27
28 do {
29 if (node === toNode) {
30 return offsetWalked +
31 toOffset - fromOffset ;
32 }
33 //offsetWalked += node .
34 nodeValue . trim () . length ;
35 offsetWalked += Type .
36 TextWalker . _textLength (
37 node) ;
38
39 } while (node = dom . next ()) ;
40 return null ;
41 } ;
42
43 /**
44 * todo constraining node
45 * @param {Node} fromNode
46 * @param {number} offset
47 * @returns {{node:Node,offset:
48 number}} null} - The node and
49 the offset to its
50 start or null if no node
51 could be found
52 */
53 Type . TextWalker . nodeAt = function
54 (fromNode, offset ,
55 startOffset) {
56
57 var walker = new Type . DomWalker (
58 fromNode, 'textual' ) ,
59 //var walker = new Type .
60 DomWalker (fromNode, 'text ' )
61
62 node = walker . first () , //Type .
63 DomWalker . first (fromNode,
64 'text ' ) ,
65 offsetWalked = 0 ,
66 length ;
67
68 startOffset = startOffset || 0 ;
69 offset += startOffset ;
70
71 //if (fromNode . nodeType === 3 &&
72 offset >= 0 && offset <=
73 fromNode . nodeValue . trim () .
74 length) {
75
76 } else {
77
78 }
79
80 }
81
82 }
83
84 }
85
86 }
87
88 }
89
90 }
91
92 }
93
94 }
95
96 }
97
98 }
99
100 }
101
102 }
103
104 }
105
106 }
107
108 }
109
110 }
111
112 }
113
114 }
115
116 }
117
118 }
119
120 }
121
122 }
123
124 }
125
126 }
127
128 }
129
130 }
131
132 }
133
134 }
135
136 }
137
138 }
139
140 }
141
142 }
143
144 }
145
146 }
147
148 }
149
150 }
151
152 }
153
154 }
155
156 }
157
158 }
159
160 }
161
162 }
163
164 }
165
166 }
167
168 }
169
170 }
171
172 }
173
174 }
175
176 }
177
178 }
179
180 }
181
182 }
183
184 }
185
186 }
187
188 }
189
190 }
191
192 }
193
194 }
195
196 }
197
198 }
199
200 }
201
202 }
203
204 }
205
206 }
207
208 }
209
210 }
211
212 }
213
214 }
215
216 }
217
218 }
219
220 }
221
222 }
223
224 }
225
226 }
227
228 }
229
230 }
231
232 }
233
234 }
235
236 }
237
238 }
239
240 }
241
242 }
243
244 }
245
246 }
247
248 }
249
250 }
251
252 }
253
254 }
255
256 }
257
258 }
259
260 }
261
262 }
263
264 }
265
266 }
267
268 }
269
270 }
271
272 }
273
274 }
275
276 }
277
278 }
279
280 }
281
282 }
283
284 }
285
286 }
287
288 }
289
290 }
291
292 }
293
294 }
295
296 }
297
298 }
299
300 }
301
302 }
303
304 }
305
306 }
307
308 }
309
310 }
311
312 }
313
314 }
315
316 }
317
318 }
319
320 }
321
322 }
323
324 }
325
326 }
327
328 }
329
330 }
331
332 }
333
334 }
335
336 }
337
338 }
339
340 }
341
342 }
343
344 }
345
346 }
347
348 }
349
350 }
351
352 }
353
354 }
355
356 }
357
358 }
359
360 }
361
362 }
363
364 }
365
366 }
367
368 }
369
370 }
371
372 }
373
374 }
375
376 }
377
378 }
379
380 }
381
382 }
383
384 }
385
386 }
387
388 }
389
390 }
391
392 }
393
394 }
395
396 }
397
398 }
399
400 }
401
402 }
403
404 }
405
406 }
407
408 }
409
410 }
411
412 }
413
414 }
415
416 }
417
418 }
419
420 }
421
422 }
423
424 }
425
426 }
427
428 }
429
430 }
431
432 }
433
434 }
435
436 }
437
438 }
439
440 }
441
442 }
443
444 }
445
446 }
447
448 }
449
450 }
451
452 }
453
454 }
455
456 }
457
458 }
459
460 }
461
462 }
463
464 }
465
466 }
467
468 }
469
470 }
471
472 }
473
474 }
475
476 }
477
478 }
479
480 }
481
482 }
483
484 }
485
486 }
487
488 }
489
490 }
491
492 }
493
494 }
495
496 }
497
498 }
499
500 }
501
502 }
503
504 }
505
506 }
507
508 }
509
510 }
511
512 }
513
514 }
515
516 }
517
518 }
519
520 }
521
522 }
523
524 }
525
526 }
527
528 }
529
530 }
531
532 }
533
534 }
535
536 }
537
538 }
539
540 }
541
542 }
543
544 }
545
546 }
547
548 }
549
550 }
551
552 }
553
554 }
555
556 }
557
558 }
559
560 }
561
562 }
563
564 }
565
566 }
567
568 }
569
570 }
571
572 }
573
574 }
575
576 }
577
578 }
579
580 }
581
582 }
583
584 }
585
586 }
587
588 }
589
590 }
591
592 }
593
594 }
595
596 }
597
598 }
599
600 }
601
602 }
603
604 }
605
606 }
607
608 }
609
610 }
611
612 }
613
614 }
615
616 }
617
618 }
619
620 }
621
622 }
623
624 }
625
626 }
627
628 }
629
630 }
631
632 }
633
634 }
635
636 }
637
638 }
639
640 }
641
642 }
643
644 }
645
646 }
647
648 }
649
650 }
651
652 }
653
654 }
655
656 }
657
658 }
659
660 }
661
662 }
663
664 }
665
666 }
667
668 }
669
670 }
671
672 }
673
674 }
675
676 }
677
678 }
679
680 }
681
682 }
683
684 }
685
686 }
687
688 }
689
690 }
691
692 }
693
694 }
695
696 }
697
698 }
699
700 }
701
702 }
703
704 }
705
706 }
707
708 }
709
710 }
711
712 }
713
714 }
715
716 }
717
718 }
719
720 }
721
722 }
723
724 }
725
726 }
727
728 }
729
730 }
731
732 }
733
734 }
735
736 }
737
738 }
739
740 }
741
742 }
743
744 }
745
746 }
747
748 }
749
750 }
751
752 }
753
754 }
755
756 }
757
758 }
759
760 }
761
762 }
763
764 }
765
766 }
767
768 }
769
770 }
771
772 }
773
774 }
775
776 }
777
778 }
779
780 }
781
782 }
783
784 }
785
786 }
787
788 }
789
790 }
791
792 }
793
794 }
795
796 }
797
798 }
799
800 }
801
802 }
803
804 }
805
806 }
807
808 }
809
810 }
811
812 }
813
814 }
815
816 }
817
818 }
819
820 }
821
822 }
823
824 }
825
826 }
827
828 }
829
830 }
831
832 }
833
834 }
835
836 }
837
838 }
839
840 }
841
842 }
843
844 }
845
846 }
847
848 }
849
850 }
851
852 }
853
854 }
855
856 }
857
858 }
859
860 }
861
862 }
863
864 }
865
866 }
867
868 }
869
870 }
871
872 }
873
874 }
875
876 }
877
878 }
879
880 }
881
882 }
883
884 }
885
886 }
887
888 }
889
890 }
891
892 }
893
894 }
895
896 }
897
898 }
899
900 }
901
902 }
903
904 }
905
906 }
907
908 }
909
910 }
911
912 }
913
914 }
915
916 }
917
918 }
919
920 }
921
922 }
923
924 }
925
926 }
927
928 }
929
930 }
931
932 }
933
934 }
935
936 }
937
938 }
939
940 }
941
942 }
943
944 }
945
946 }
947
948 }
949
950 }
951
952 }
953
954 }
955
956 }
957
958 }
959
960 }
961
962 }
963
964 }
965
966 }
967
968 }
969
970 }
971
972 }
973
974 }
975
976 }
977
978 }
979
980 }
981
982 }
983
984 }
985
986 }
987
988 }
989
990 }
991
992 }
993
994 }
995
996 }
997
998 }
999
1000 }

```

```

82         offset) {
83             return { node: node,
84                     offset: offset -
85                     offsetWalked };
86         }
87         offsetWalked += length;
88         } while (node = walker.next())
89         ;
90         return null;
91     };
92     Type.TextWalker._textLength =
function (node) {
93         if (node.nodeName.
94             toLocaleLowerCase() === 'br
95             ') {
96             return 1;
97         } else {
98             return node.nodeValue.trim().
99             length;
100         }
101     };
102     /**
103     * @param a
104     * @param b
105     */
106     this.mergeTexts = function (a, b
107     ) {
108         if (a.nodeType === Node.
109         TEXT_NODE) {
110             //
111             //
112             //
113             //
114             //
115             //
116             //
117             //
118             //
119             //
120             //
121             //
122             //
123             //
124             //
125             //
126             //
127             //
128             //
129             //
130             //
131             //
132             //
133             //
134             //
135             //
136             //
137             //
138             //
139             //
140             //
141             //
142             //
143             //
144             //
145             //
146             //
147             //
148             //
149             //
150             //
151             //
152             //
153             //
154             //
155             //
156             //
157             //
158             //
159             //
160             //
161             //
162             //
163             //
164             //
165             //
166             //
167             //
168             //
169             //
170             //
171             //
172             //
173             //
174             //
175             //
176             //
177             //
178             //
179             //
180             //
181             //
182             //
183             //
184             //
185             //
186             //
187             //
188             //
189             //
190             //
191             //
192             //
193             //
194             //
195             //
196             //
197             //
198             //
199             //
200             //
201             //
202             //
203             //
204             //
205             //
206             //
207             //
208             //
209             //
210             //
211             //
212             //
213             //
214             //
215             //
216             //
217             //
218             //
219             //
220             //
221             //
222             //
223             //
224             //
225             //
226             //
227             //
228             //
229             //
230             //
231             //
232             //
233             //
234             //
235             //
236             //
237             //
238             //
239             //
240             //
241             //
242             //
243             //
244             //
245             //
246             //
247             //
248             //
249             //
250             //
251             //
252             //
253             //
254             //
255             //
256             //
257             //
258             //
259             //
260             //
261             //
262             //
263             //
264             //
265             //
266             //
267             //
268             //
269             //
270             //
271             //
272             //
273             //
274             //
275             //
276             //
277             //
278             //
279             //
280             //
281             //
282             //
283             //
284             //
285             //
286             //
287             //
288             //
289             //
290             //
291             //
292             //
293             //
294             //
295             //
296             //
297             //
298             //
299             //
300             //
301             //
302             //
303             //
304             //
305             //
306             //
307             //
308             //
309             //
310             //
311             //
312             //
313             //
314             //
315             //
316             //
317             //
318             //
319             //
320             //
321             //
322             //
323             //
324             //
325             //
326             //
327             //
328             //
329             //
330             //
331             //
332             //
333             //
334             //
335             //
336             //
337             //
338             //
339             //
340             //
341             //
342             //
343             //
344             //
345             //
346             //
347             //
348             //
349             //
350             //
351             //
352             //
353             //
354             //
355             //
356             //
357             //
358             //
359             //
360             //
361             //
362             //
363             //
364             //
365             //
366             //
367             //
368             //
369             //
370             //
371             //
372             //
373             //
374             //
375             //
376             //
377             //
378             //
379             //
380             //
381             //
382             //
383             //
384             //
385             //
386             //
387             //
388             //
389             //
390             //
391             //
392             //
393             //
394             //
395             //
396             //
397             //
398             //
399             //
400             //
401             //
402             //
403             //
404             //
405             //
406             //
407             //
408             //
409             //
410             //
411             //
412             //
413             //
414             //
415             //
416             //
417             //
418             //
419             //
420             //
421             //
422             //
423             //
424             //
425             //
426             //
427             //
428             //
429             //
430             //
431             //
432             //
433             //
434             //
435             //
436             //
437             //
438             //
439             //
440             //
441             //
442             //
443             //
444             //
445             //
446             //
447             //
448             //
449             //
450             //
451             //
452             //
453             //
454             //
455             //
456             //
457             //
458             //
459             //
460             //
461             //
462             //
463             //
464             //
465             //
466             //
467             //
468             //
469             //
470             //
471             //
472             //
473             //
474             //
475             //
476             //
477             //
478             //
479             //
480             //
481             //
482             //
483             //
484             //
485             //
486             //
487             //
488             //
489             //
490             //
491             //
492             //
493             //
494             //
495             //
496             //
497             //
498             //
499             //
500             //
501             //
502             //
503             //
504             //
505             //
506             //
507             //
508             //
509             //
510             //
511             //
512             //
513             //
514             //
515             //
516             //
517             //
518             //
519             //
520             //
521             //
522             //
523             //
524             //
525             //
526             //
527             //
528             //
529             //
530             //
531             //
532             //
533             //
534             //
535             //
536             //
537             //
538             //
539             //
540             //
541             //
542             //
543             //
544             //
545             //
546             //
547             //
548             //
549             //
550             //
551             //
552             //
553             //
554             //
555             //
556             //
557             //
558             //
559             //
560             //
561             //
562             //
563             //
564             //
565             //
566             //
567             //
568             //
569             //
570             //
571             //
572             //
573             //
574             //
575             //
576             //
577             //
578             //
579             //
580             //
581             //
582             //
583             //
584             //
585             //
586             //
587             //
588             //
589             //
590             //
591             //
592             //
593             //
594             //
595             //
596             //
597             //
598             //
599             //
600             //
601             //
602             //
603             //
604             //
605             //
606             //
607             //
608             //
609             //
610             //
611             //
612             //
613             //
614             //
615             //
616             //
617             //
618             //
619             //
620             //
621             //
622             //
623             //
624             //
625             //
626             //
627             //
628             //
629             //
630             //
631             //
632             //
633             //
634             //
635             //
636             //
637             //
638             //
639             //
640             //
641             //
642             //
643             //
644             //
645             //
646             //
647             //
648             //
649             //
650             //
651             //
652             //
653             //
654             //
655             //
656             //
657             //
658             //
659             //
660             //
661             //
662             //
663             //
664             //
665             //
666             //
667             //
668             //
669             //
670             //
671             //
672             //
673             //
674             //
675             //
676             //
677             //
678             //
679             //
680             //
681             //
682             //
683             //
684             //
685             //
686             //
687             //
688             //
689             //
690             //
691             //
692             //
693             //
694             //
695             //
696             //
697             //
698             //
699             //
700             //
701             //
702             //
703             //
704             //
705             //
706             //
707             //
708             //
709             //
710             //
711             //
712             //
713             //
714             //
715             //
716             //
717             //
718             //
719             //
720             //
721             //
722             //
723             //
724             //
725             //
726             //
727             //
728             //
729             //
730             //
731             //
732             //
733             //
734             //
735             //
736             //
737             //
738             //
739             //
740             //
741             //
742             //
743             //
744             //
745             //
746             //
747             //
748             //
749             //
750             //
751             //
752             //
753             //
754             //
755             //
756             //
757             //
758             //
759             //
760             //
761             //
762             //
763             //
764             //
765             //
766             //
767             //
768             //
769             //
770             //
771             //
772             //
773             //
774             //
775             //
776             //
777             //
778             //
779             //
780             //
781             //
782             //
783             //
784             //
785             //
786             //
787             //
788             //
789             //
790             //
791             //
792             //
793             //
794             //
795             //
796             //
797             //
798             //
799             //
800             //
801             //
802             //
803             //
804             //
805             //
806             //
807             //
808             //
809             //
810             //
811             //
812             //
813             //
814             //
815             //
816             //
817             //
818             //
819             //
820             //
821             //
822             //
823             //
824             //
825             //
826             //
827             //
828             //
829             //
830             //
831             //
832             //
833             //
834             //
835             //
836             //
837             //
838             //
839             //
840             //
841             //
842             //
843             //
844             //
845             //
846             //
847             //
848             //
849             //
850             //
851             //
852             //
853             //
854             //
855             //
856             //
857             //
858             //
859             //
860             //
861             //
862             //
863             //
864             //
865             //
866             //
867             //
868             //
869             //
870             //
871             //
872             //
873             //
874             //
875             //
876             //
877             //
878             //
879             //
880             //
881             //
882             //
883             //
884             //
885             //
886             //
887             //
888             //
889             //
890             //
891             //
892             //
893             //
894             //
895             //
896             //
897             //
898             //
899             //
900             //
901             //
902             //
903             //
904             //
905             //
906             //
907             //
908             //
909             //
910             //
911             //
912             //
913             //
914             //
915             //
916             //
917             //
918             //
919             //
920             //
921             //
922             //
923             //
924             //
925             //
926             //
927             //
928             //
929             //
930             //
931             //
932             //
933             //
934             //
935             //
936             //
937             //
938             //
939             //
940             //
941             //
942             //
943             //
944             //
945             //
946             //
947             //
948             //
949             //
950             //
951             //
952             //
953             //
954             //
955             //
956             //
957             //
958             //
959             //
960             //
961             //
962             //
963             //
964             //
965             //
966             //
967             //
968             //
969             //
970             //
971             //
972             //
973             //
974             //
975             //
976             //
977             //
978             //
979             //
980             //
981             //
982             //
983             //
984             //
985             //
986             //
987             //
988             //
989             //
990             //
991             //
992             //
993             //
994             //
995             //
996             //
997             //
998             //
999             //
1000            //

```

Listing 11: text_walker.js

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6   * Crates a new Type.Range
7   *
8   * Type.Range is a shim for the
9   * browsers' native {Range}
10  * objects and
11  * is being used in Type for
12  * anything related to text
13  * ranges.
14
15  * Native ranges are often buggy,
16  * lack essential features and
17  * should
18  * not be used other than for
19  * performance reasons. This
20  * class avoids
21  * and / or fixes common issues with
22  * ranges and adds many methods
23  * useful for text editing.
24
25  * Among many other factory methods,
26  * you can use the {Type.Range.
27  * fromRange}
28
29
30  this.startOffset = startOffset;
31  this.endContainer = endContainer;
32  this.endOffset = endOffset;
33
34  this.
35  ensureStartNodePrecedesEndNode
36  ();
37
38  (function () {
39
40  /**
41  * If the startContainer and the
42  * endContainer are enclosed by
43  * the same element matching the
44  * selector, that element will
45  * be returned. Otherwise null
46  * will be returned.
47
48  * todo call this commonAncestor
49  * and make the selector
50  * optional
51
52  * @param {String} selector - This
53  * method will only return a

```



```

134         and set
135         * this' startContainer to the
136           right node the resulting
137           nodes of the
138         * split and the startOffset to 0.
139         Will return the new
140         startContainer.
141         * @returns {Node} - The new
142         startContainer
143         */
144         this.splitStartContainer =
145         function () {
146             var startsAndEndsInSameNode;
147             if (this.startOffset === 0) {
148                 return this.startContainer;
149             }
150             startsAndEndsInSameNode = this.
151             startsAndEndsInSameNode();
152             this.startContainer = this.
153             startContainer.splitText(
154                 this.startOffset);
155             if (startsAndEndsInSameNode) {
156                 this.endContainer = this.
157                 startContainer;
158                 this.endOffset -= this.
159                 startOffset;
160             }
161             this.startOffset = 0;
162             return this.startContainer;
163         };
164         /** Will split the endContainer
165         text node at the endOffset
166         and set
167         * this' endContainer to the left
168         node the resulting nodes of
169         the
170         * split and the endOffset to the
171         end of the endContainer.
172         Will return the new
173         endContainer.
174         * @returns {Node} - The new
175         endContainer
176         */
177         this.splitEndContainer = function
178         () {
179             if (this.endOffset !== this.
180             endContainer.length) {
181                 this.endContainer = this.
182                 endContainer.splitText(
183                     this.endOffset).
184                 previousSibling;
185                 this.endOffset = this.
186                 endContainer.length;
187             }
188             return this.endContainer;
189         };
190         /** Creates a native {Range} object
191         and returns it.
192         * @returns {Range}
193         */
194         this.getNativeRange = function ()
195         {
196             var range = document.createRange
197             ();
198             range.setEnd(this.endContainer,
199             this.endOffset);
200             range.setStart(this.
201             startContainer, this.
202             startOffset);
203             return range;
204         };
205         /** Looks up the number of
206         characters (offsets) where
207         this range starts
208         * and ends relative to a given {
209         Element}. Returns an {Object}
210         containing
211         }
212         * Returns the offset (number of
213         visible characters) from the
214         given node
215         * to the startContainer and its
216         startOffset. If no node has
217         been passed
218         * this will return the
219         startOffset
220

```

```

215 * @param {Node} [from] - The node
216 * to start counting
217 * characters from
218 * @returns {number| null}
219 */
220 this.getStartOffset = function (
221   from) {
222   if (from) {
223     return Type.TextWalker.offset(
224       from, this.startContainer
225       , 0, this.startOffset);
226   }
227   return parseInt(this.startOffset
228     , 10);
229 }
230 /**
231 * Returns the offset (number of
232 * visible characters) from the
233 * given node
234 * to the endContainer and its
235 * endOffset. If no node has
236 * been passed
237 * this will return the endOffset
238 * @param {Node} [from] - The node
239 * to start counting
240 * characters from
241 * @returns {number| null}
242 */
243 this.getEndOffset = function (from
244 ) {
245   if (from) {
246     return Type.TextWalker.offset(
247       from, this.endContainer,
248       0, this.endOffset);
249   }
250   return parseInt(this.endOffset,
251     10);
252 }
253 /**
254 * Returns whether or not the the
255 * element containing the
256 * startContainer is of the given
257 * tagName.
258 * @param {string} tagName - The
259 * tag name to compare.
260 * @returns {boolean}
261 */
262 this.startTagsIs = function (
263   tagName) {
264   return this.getStartTagName()
265     === tagName.toLowerCase();
266 }
267 /**
268 * Returns whether or not the the
269 * element containing the
270 * endContainer is of the given
271 * tagName.
272 * @param {string} tagName - The
273 * tag name to compare.
274 * @returns {boolean}
275 */
276 this.endTagsIs = function (tagName)
277 {
278   return this.getEndTagName() ===
279     tagName.toLowerCase();
280 }
281 /**
282 * Returns whether or not the the
283 * element containing the
284 * startContainer equals the
285 * endContainer.
286 * @param {boolean}
287 */
288 this.startsAndEndsInSameNode =
289   function () {
290     return this.startContainer ===
291       this.endContainer;
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }

```

```

310 /**
311  * Returns whether or not this
312  * range spans over no
313  * characters
314  * at all.
315  * @returns {boolean}
316  */
317 this.isCollapsed = function () {
318   return this.startOffset === this
     .endOffset && this
     .startsAndEndsInSameNode();
319 };
320 /**
321  * Merges another range with this
322  * range and returns this range
323  * .
324  * @param {Type.Range} that - The
     range that should be added
     to
325  * this range.
326  * @returns {Type.Range} - This
     instance
327  */
328 this.mergeWith = function (that) {
329   var startOrder, endOrder;
330   startOrder = Type.DomUtilities.
331     order(this.startContainer,
332       that.startContainer);
333   endOrder = Type.DomUtilities.
334     order(this.endContainer,
335       that.endContainer);
336   if (startOrder === 0) {
337     this.startOffset = Math.min(
338       this.startOffset, that.
339       startOffset);
340   } else if (startOrder === 1) {
341     this.startContainer = that.
342     startContainer;
343   }
344   if (endOrder === 0) {
345     this.endOffset = Math.max(this
346       .endOffset, that.
347       endOffset);
348   } else if (endOrder === -1) {
349     this.endContainer = that.
350     endContainer;
351   }
352   return this;
353 }
354 /**
355  * Internal method to swap the
356  * start and end containers as
357  * well
358  * as their offsets when it is
359  * initialized with the
360  * endContainer
361  * preceding the startContainer.
362  * @returns {Type.Range} - This
     instance
363  * @private
364  */
365 this._swapStartAndEnd = function
     () {
366   this._swapContainers();
367   this._swapOffsets();
368   return this;
369 };
370 /**
371  * Will swap the startContainer
372  * with the endContainer
373  * @returns {Type.Range} - This
     instance
374  * @private
375  */
376 this._swapContainers = function () {
377   var swapContainers = function () {
378     var swapContainer = this.
379     startContainer;
380     this.startContainer = this.
381     endContainer;
382     this.endContainer = swapContainer;
383   };
384   swapContainers();
385 }
386 /**
387  * Will swap the startOffset with
388  * the endOffset
389  * @returns {Type.Range} - This
     instance
390  * @private
391  */
392 this._swapOffsets = function () {
393   var swapOffset = this.
394   startOffset;
395   this.startOffset = this.
396   endOffset;
397   this.endOffset = swapOffset;
398   return this;
399 };
400 /**
401  * call (Type.Range.prototype);
402  */
403 (function () {
404   /**
405    * The implementation of {Range#
406    * getClientRects} is broken in
407    * WebKit
408    * browsers. {@link Type.Range.
409    * _getClientRectsNeedsFix}
410    * tests for
411    * wrong behaviour and stores if
412    * it is broken in this
413    * variable.
414    * @type {null|boolean}
415    */
416   Type.Range._getClientRectsIsBroken
     = null;
417 }
418 )();
419 
```



```

474         if (endOffset === 0 &&
475             endContainer === Type.
476                 DomWalker.next(range.
477                     startContainer.parentNode.
478                     nextSibling, 'visible')) {
479             endContainer = Type.DomWalker.
480                 last(range.startContainer
481                     .parentNode, 'text');
482             endOffset = endContainer.
483                 length;
484         }
485         return new Type.Range(range.
486             startContainer, range.
487             startOffset, endContainer,
488             endOffset);
489     };
490     /** Will create a {Type.Range}
491         spanning from the offset of
492         the given {Caret}
493         over a number of characters
494         passed as selectedChars. If
495         selectedChars is
496         a positive number, the range's
497         start will be set to the
498         cursor position
499         and the end spanning to the
500         characters to its right. If
501         selectedChars is
502         negative it will span to the
503         characters to its left.
504     */
505     * @param {Caret} caret
506     * @param {number} selectedChars
507     * @returns {Type.Range}
508     */
509     Type.Range.fromCaret = function (
510         caret, selectedChars) {
511         var startNode = caret.getNode(),
512             startOffset = caret.
513                 getNodeOffset(),
514             end = Type.TextWalker.nodeAt(
515                 startNode, selectedChars,
516                 startOffset);
517         return new Type.Range(startNode,
518             end,
519             startNode,
520             endOffset);
521     };
522     /** Will create a {Type.Range} at
523         the offset and inside the
524         text node
525         found at the x and y positions
526         relative to the document.
527         The range
528         will be collapsed. Will return
529         null
530     */
531     * @param {number} x - The
532         horizontal position relative
533         to the document
534     * @param {number} y - The
535         vertical position relative
536         to the document
537     * @returns {Type.Range|null} -
538         Returns a new Type.Range or
539         null if the
540         position is not inside a
541         text node
542     */
543     Type.Range.fromPoint = function (x
544         , y) {
545         var range, node, offset;
546         if (document.
547             caretPositionFromPoint) {
548             range = document.
549 
```

```

545         caretPositionFromPoint(x,
546         y);
547         node = range.offsetNode;
548         offset = range.offset;
549     } else if (document.
550     caretRangeFromPoint) {
551         range = document.
552         caretRangeFromPoint(x, y)
553         ;
554         node = range.startContainer;
555         offset = range.startOffset;
556     } else {
557         Type.Development.debug('This
558         browser does not support
559         caretPositionFromPoint or
560         caretRangeFromPoint. ');
561         return null;
562     }
563     // only split TEXT_NODES
564     if (node.nodeType === Node.
565     TEXT_NODE) {
566         return new Type.Range(node,
567         offset, node, offset);
568     }
569     Type.Development.debug('User
570     clicked in a non-text node,
571     cannot create range');
572     return null;
573 };
574 /**
575  * WebKit browsers sometimes
576  * create unnecessary and
577  * overlapping {ClientRect}s in
578  * {Range.prototype.getClientRects
579  * }. This method creates 2
580  * elements, creates a
581  * range and tests for this
582  * behaviour.
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

631 * @return {ClientRect[]}
632 *   ClientRectList or list of
633 *   ClientRect objects
634 *   describing range
635 *
636 Type.Range.getClientRects =
  function (range) {
637   if (!Type.Range.
638     _getClientRectsNeedsFix())
639     return range.getClientRects();
640   var partialRange = document.
641     createRange(), range.
642     endContainer = range.
643     endOffset = range.endOffset,
644     rects = [];
645   while (endContainer != range.
646     commonAncestorContainer) {
647     partialRange.setStart(
648       endContainer, 0);
649     partialRange.setEnd(
650       endContainer, endOffset);
651     Array.prototype.push.apply(
652       rects, partialRange.
653       getClientRects());
654     endOffset = Array.prototype.
655       indexOf.call(
656         endContainer.parentNode.
657         childNodes, endContainer
658       );
659     endContainer = endContainer.
660       parentNode;
661   }
662   return rects;
663 }
664
665 }).call();
666
667 module.exports = Type.Range;

```

Listing 12: range.js

```

1 'use strict';
2
3 var Type = require('./core');
4
5 /**
6 *
7 * @param {Type} type
8 * @constructor
9 *
10 Type.Writer = function (type) {
11   this._type = type;
12   this._root = type.getRoot();
13 };
14
15 (function () {
16   /**
17    * Inserts a string in a text node
18    * at a given offset
19    *
20    * @param {Text} textNode - The
21
22   text node into which str
23   will be inserted.
24   * @param {Number} offset - The
25   character offset at which
26   str will be inserted.
27   * @param {String} str - The text
28   that will be inserted
29   * @returns {Type.Writer} - This
30   instance
31   */
32   this.insertText = function (
33     textNode, offset, str) {
34     var nodeText = textNode.
35       nodeValue;
36     if (offset > 0) {
37       textNode.nodeValue = nodeText.
38         substring(0, offset) +
39         str + nodeText.substring(
40           offset, nodeText.length);
41     } else {
42
43       textNode.nodeValue = str +
44         nodeText;
45       return this;
46     };
47   }
48   /**
49    * Inserts DOM nodes at the offset
50    * of a text node
51    *
52    * @param {Text} textNode - The
53    text node which will be
54    split and in which
55    the DOM will be inserted.
56    * @param {Number} offset - The
57    text offset at which the DOM
58    should be
59    inserted.
60    * @param {Node||NodeList|
61    String} nodes - Either a {
62    Node}, an array

```

```

46 *   of {Node}s, a {NodeList} or
47 *   a string containing HTML
48 *   that will be
49 *   inserted at the given
50 *   offset in a text node.
51 * @returns {Type.Writer} - This
52 * instance
53 */
54 this.insertHTML = function (
55   textNode, offset, nodes) {
56
57   // Required variables
58   var i, parent, insertBeforeNode;
59
60   // Parse string (if given) to
61   // retrieve DOM nodes
62   nodes = typeof nodes === 'string'
63     ? Type.DomUtilities.
64     parseHTML(nodes) : nodes;
65
66   // Make array if single DOM node
67   // was given
68   nodes = nodes.length ? nodes : [
69     nodes];
70
71   // Make nodes an array (in case
72   // it is a NodeList)
73   nodes = Array.prototype.slice.
74     call(nodes);
75
76   // Split text and prepare
77   // insertion
78   insertBeforeNode = textNode.
79     splitText(offset);
80   parent = insertBeforeNode.
81     parentNode;
82
83   // If last given DOM node is a
84   // text, concat it with the
85   // text, behind insertion
86   if (nodes[nodes.length - 1].
87     nodeType === Node.TEXT_NODE
88   ) {
89     insertBeforeNode.nodeValue =
90       nodes.pop().nodeValue +
91       insertBeforeNode.
92
93   of {Node}s, a {NodeList} or
94   a string containing HTML
95   that will be
96   inserted at the given
97   offset in a text node.
98   @returns {Type.Writer} - This
99   instance
100 */
101 this.remove = function (range,
102   numChars) {
103
104   //var startNode, endNode,
105   //startParent, walker,
106   //current, prev, startRemoved,
107   //currentParent, a, b;
108
109   var startNode, endNode, walker,
110     current, prev, startRemoved,
111     currentParent, a, b;
112
113   if (arguments.length === 2) {
114     range = Type.Range.fromCaret(
115       range, numChars);
116   }
117
118   startNode = range.
119     splitStartContainer();
120   endNode = range.
121     splitEndContainer();
122   startParent = startNode.
123     parentNode;
124   walker = this._type.
125     createDomWalker(endNode,
126       textNode);
127   //current = endNode;
128   startRemoved = false;
129
130   //prev = endNode;
131
132   if (!this._root.contains(
133     startNode) || !this._root.
134     contains(endNode)) {
135     Type.Development.debug('The
136       give startNode and
137       endNode are not contained

```

```

127         by the editor.');
128         return this;
129     }
130     while (!startRemoved) {
131         current = walker.getNode();
132         walker.prev();
133         a = (current === endNode &&
134             range.endOffset === 0);
135         b = (current !== startNode &&
136             current === Type.
137             DomWalker.first(current,
138                 parentNode, 'textNode'));
139         if (a || b) {
140             currentParent = current.
141                 parentNode;
142             Type.DomUtilities.moveAfter(
143                 walker.getNode(),
144                 current.parentNode.
145                 childNodes);
146             Type.DomUtilities.
147                 removeVisible(
148                     currentParent);
149         }
150     }
151     'use strict';
152     var Type = require('./core');
153     /**
154     * @param {Type} type
155     * @constructor
156     */
157     Type.Formatter = function (type) {
158         this._type = type;
159     };
160     (function () {
161         startRemoved = current ===
162             startNode;
163         Type.DomUtilities.
164             removeVisible(current);
165         //current = walker.getNode();
166     }
167     /*while (!startRemoved) {
168         prev = Type.DomWalker.prev(
169             current, 'text');
170         a = (current === endNode &&
171             range.endOffset === 0);
172         b = (current !== startNode &&
173             current === Type.
174             DomWalker.first(current,
175                 parentNode, 'text'));
176         if (a || b) {
177             currentParent = current.
178                 parentNode;
179             Type.DomUtilities.moveAfter(
180                 prev, current,
181                 parentNode.childNodes);
182         }
183     }
184     */
185     this._inlineTags = ["strong", "em",
186         "u", "s"];
187     /**
188     * A list of tags that are
189     * displayed as block elements.
190     * We generate different
191     * markup for inline and block
192     * tags. We use this array as
193     * reference to determine
194     * what kind of markup to generate
195     */
196     * todo move me to dom utils
197     * @type {string[]}
198     * @private
199 
```

Listing 13: writer.js

```

34 * @type {string[]}
35 * @private
36 */
37 this._blockTags = ["h1", "h2", "
38 h3", "h4", "h5", "h6", "
39 blockquote"];
40
41 /**
42 * Will call either this.inline,
43 * this.block or this._noop
44 * depending on
45 * whether the given tag is an
46 * inline or block element or
47 * we do not know
48 * this tag yet (the latter would
49 * call _noop which would utter
50 * no action).
51
52 * @param {String} tag - The tag
53 * that we want to format the
54 * text with
55 * @param {Type.Range} typeRange -
56 * An object containing data
57 * on which part
58 * of the text to format
59 * @param {...*} params - Any
60 * number of arguments that
61 * specify attributes
62 * for the tag
63 * @returns {Element[]} - The
64 * elements created by the
65 * formatting function
66
67 this.format = function (tag,
68 typeRange, params) {
69 typeRange.ensureIsInside(this._
70 type.getRoot());
71 return this._handlerFor(tag).
72 apply(this, arguments);
73 };
74
75 /**
76 * @param tag
77 * @param typeRange
78 * @param params
79 * @returns {Type.Formatter|
80 Element[]}
81
82 this.inline = function (tag,
83 typeRange, params) {
84 var args, startNode, endNode,
85 enclosingTag, selfPositions;
86
87 // If the selection is enclosed
88 // the tag we want to format
89 // with
90 // remove formatting from
91
92 * @param range
93 * @returns {*}
94 */
95 this.removeFormat = function (tag,
96 range) {
97 var startNode = this._
98 _getStartNode(tag, range),
99 dom = this._type.
100 createDomWalker(startNode
101 ),
102 next;
103 do {
104 Type.DomUtilities.removeTag(
105 dom.getNode(), tag, false
106 );
107 next = dom.next();
108 } while(next && !next.contains(
109 range.endContainer)); // !==
110 range.endContainer);
111
112 return this;
113 };
114
115 /**
116 * @param tag
117 * @param typeRange
118 * @param params
119 * @returns {Type.Formatter|
120 Element[]}
121
122 this.inline = function (tag,
123 typeRange, params) {
124 var args, startNode, endNode,
125 enclosingTag, selfPositions;
126
127 // If the selection is enclosed
128 // the tag we want to format
129 // with
130 // remove formatting from
131
132 * @param range
133 * @returns {*}
134 */
135 this.removeFormat = function (tag,
136 range) {
137 var startNode = this._
138 _getStartNode(tag, range),
139 dom = this._type.
140 createDomWalker(startNode
141 ),
142 next;
143 do {
144 Type.DomUtilities.removeTag(
145 dom.getNode(), tag, false
146 );
147 next = dom.next();
148 } while(next && !next.contains(
149 range.endContainer)); // !==
150 range.endContainer);
151
152 return this;
153 };
154
155 /**
156 * This method will wrap the given
157 * tag around (and including)
158 * all elements
159 * between the startNode and
160 * endNode and try to maintain
161 * simple and valid
162 * HTML. The tag should be an "
163 * inline"-element, for "block"
164 * elements use
165 * {block}. Both methods have a
166 * different behaviour when
167 * generating markup.
168
169 * @param {String} tag
170 * @param {Node} startNode
171 * @param {Node} endNode
172 * @param {...*} [params]
173 * @returns {Element[]} - The
174 * elements created by the
175 * formatting function
176
177 selected area
178 if (enclosingTag = typeRange.
179 elementEnclosingStartAndEnd
180 (tag)) {
181 return this.removeInline(
182 enclosingTag, typeRange);
183 }
184 // Otherwise add formatting to
185 selected area
186 } else {
187 startNode = this._getStartNode
188 (tag, typeRange);
189 endNode = this._getEndNode(
190 tag, typeRange);
191 params = Array.prototype.
192 slice.call(arguments, 2);
193 args = [tag, startNode,
194 endNode].concat(params);
195 return this.insertInline.apply
196 (this, args);
197 }
198
199 }
200
201 };
202
203 /**
204 * This method will wrap the given
205 * tag around (and including)
206 * all elements
207 * between the startNode and
208 * endNode and try to maintain
209 * simple and valid
210 * HTML. The tag should be an "
211 * inline"-element, for "block"
212 * elements use
213 * {block}. Both methods have a
214 * different behaviour when
215 * generating markup.
216
217 * @param {String} tag
218 * @param {Node} startNode
219 * @param {Node} endNode
220 * @param {...*} [params]
221 * @returns {Element[]} - The
222 * elements created by the
223 * formatting function
224
225 selected area
226 if (enclosingTag = typeRange.
227 elementEnclosingStartAndEnd
228 (tag)) {
229 return this.removeInline(
230 enclosingTag, typeRange);
231 }
232 // Otherwise add formatting to
233 selected area
234 } else {
235 startNode = this._getStartNode
236 (tag, typeRange);
237 endNode = this._getEndNode(
238 tag, typeRange);
239 params = Array.prototype.
240 slice.call(arguments, 2);
241 args = [tag, startNode,
242 endNode].concat(params);
243 return this.insertInline.apply
244 (this, args);
245 }
246
247 }
248
249 };
250
251 /**
252 * This method will wrap the given
253 * tag around (and including)
254 * all elements
255 * between the startNode and
256 * endNode and try to maintain
257 * simple and valid
258 * HTML. The tag should be an "
259 * inline"-element, for "block"
260 * elements use
261 * {block}. Both methods have a
262 * different behaviour when
263 * generating markup.
264
265 * @param {String} tag
266 * @param {Node} startNode
267 * @param {Node} endNode
268 * @param {...*} [params]
269 * @returns {Element[]} - The
270 * elements created by the
271 * formatting function
272
273 selected area
274 if (enclosingTag = typeRange.
275 elementEnclosingStartAndEnd
276 (tag)) {
277 return this.removeInline(
278 enclosingTag, typeRange);
279 }
280 // Otherwise add formatting to
281 selected area
282 } else {
283 startNode = this._getStartNode
284 (tag, typeRange);
285 endNode = this._getEndNode(
286 tag, typeRange);
287 params = Array.prototype.
288 slice.call(arguments, 2);
289 args = [tag, startNode,
290 endNode].concat(params);
291 return this.insertInline.apply
292 (this, args);
293 }
294
295 }
296
297 };
298
299 /**
300 * This method will wrap the given
301 * tag around (and including)
302 * all elements
303 * between the startNode and
304 * endNode and try to maintain
305 * simple and valid
306 * HTML. The tag should be an "
307 * inline"-element, for "block"
308 * elements use
309 * {block}. Both methods have a
310 * different behaviour when
311 * generating markup.
312
313 * @param {String} tag
314 * @param {Node} startNode
315 * @param {Node} endNode
316 * @param {...*} [params]
317 * @returns {Element[]} - The
318 * elements created by the
319 * formatting function
320
321 selected area
322 if (enclosingTag = typeRange.
323 elementEnclosingStartAndEnd
324 (tag)) {
325 return this.removeInline(
326 enclosingTag, typeRange);
327 }
328 // Otherwise add formatting to
329 selected area
330 } else {
331 startNode = this._getStartNode
332 (tag, typeRange);
333 endNode = this._getEndNode(
334 tag, typeRange);
335 params = Array.prototype.
336 slice.call(arguments, 2);
337 args = [tag, startNode,
338 endNode].concat(params);
339 return this.insertInline.apply
340 (this, args);
341 }
342
343 }
344
345 };
346
347 /**
348 * This method will wrap the given
349 * tag around (and including)
350 * all elements
351 * between the startNode and
352 * endNode and try to maintain
353 * simple and valid
354 * HTML. The tag should be an "
355 * inline"-element, for "block"
356 * elements use
357 * {block}. Both methods have a
358 * different behaviour when
359 * generating markup.
360
361 * @param {String} tag
362 * @param {Node} startNode
363 * @param {Node} endNode
364 * @param {...*} [params]
365 * @returns {Element[]} - The
366 * elements created by the
367 * formatting function
368
369 selected area
370 if (enclosingTag = typeRange.
371 elementEnclosingStartAndEnd
372 (tag)) {
373 return this.removeInline(
374 enclosingTag, typeRange);
375 }
376 // Otherwise add formatting to
377 selected area
378 } else {
379 startNode = this._getStartNode
380 (tag, typeRange);
381 endNode = this._getEndNode(
382 tag, typeRange);
383 params = Array.prototype.
384 slice.call(arguments, 2);
385 args = [tag, startNode,
386 endNode].concat(params);
387 return this.insertInline.apply
388 (this, args);
389 }
390
391 }
392
393 };
394
395 /**
396 * This method will wrap the given
397 * tag around (and including)
398 * all elements
399 * between the startNode and
400 * endNode and try to maintain
401 * simple and valid
402 * HTML. The tag should be an "
403 * inline"-element, for "block"
404 * elements use
405 * {block}. Both methods have a
406 * different behaviour when
407 * generating markup.
408
409 * @param {String} tag
410 * @param {Node} startNode
411 * @param {Node} endNode
412 * @param {...*} [params]
413 * @returns {Element[]} - The
414 * elements created by the
415 * formatting function
416
417 selected area
418 if (enclosingTag = typeRange.
419 elementEnclosingStartAndEnd
420 (tag)) {
421 return this.removeInline(
422 enclosingTag, typeRange);
423 }
424 // Otherwise add formatting to
425 selected area
426 } else {
427 startNode = this._getStartNode
428 (tag, typeRange);
429 endNode = this._getEndNode(
430 tag, typeRange);
431 params = Array.prototype.
432 slice.call(arguments, 2);
433 args = [tag, startNode,
434 endNode].concat(params);
435 return this.insertInline.apply
436 (this, args);
437 }
438
439 }
440
441 };
442
443 /**
444 * This method will wrap the given
445 * tag around (and including)
446 * all elements
447 * between the startNode and
448 * endNode and try to maintain
449 * simple and valid
450 * HTML. The tag should be an "
451 * inline"-element, for "block"
452 * elements use
453 * {block}. Both methods have a
454 * different behaviour when
455 * generating markup.
456
457 * @param {String} tag
458 * @param {Node} startNode
459 * @param {Node} endNode
460 * @param {...*} [params]
461 * @returns {Element[]} - The
462 * elements created by the
463 * formatting function
464
465 selected area
466 if (enclosingTag = typeRange.
467 elementEnclosingStartAndEnd
468 (tag)) {
469 return this.removeInline(
470 enclosingTag, typeRange);
471 }
472 // Otherwise add formatting to
473 selected area
474 } else {
475 startNode = this._getStartNode
476 (tag, typeRange);
477 endNode = this._getEndNode(
478 tag, typeRange);
479 params = Array.prototype.
480 slice.call(arguments, 2);
481 args = [tag, startNode,
482 endNode].concat(params);
483 return this.insertInline.apply
484 (this, args);
485 }
486
487 }
488
489 };
490
491 /**
492 * This method will wrap the given
493 * tag around (and including)
494 * all elements
495 * between the startNode and
496 * endNode and try to maintain
497 * simple and valid
498 * HTML. The tag should be an "
499 * inline"-element, for "block"
500 * elements use
501 * {block}. Both methods have a
502 * different behaviour when
503 * generating markup.
504
505 * @param {String} tag
506 * @param {Node} startNode
507 * @param {Node} endNode
508 * @param {...*} [params]
509 * @returns {Element[]} - The
510 * elements created by the
511 * formatting function
512
513 selected area
514 if (enclosingTag = typeRange.
515 elementEnclosingStartAndEnd
516 (tag)) {
517 return this.removeInline(
518 enclosingTag, typeRange);
519 }
520 // Otherwise add formatting to
521 selected area
522 } else {
523 startNode = this._getStartNode
524 (tag, typeRange);
525 endNode = this._getEndNode(
526 tag, typeRange);
527 params = Array.prototype.
528 slice.call(arguments, 2);
529 args = [tag, startNode,
530 endNode].concat(params);
531 return this.insertInline.apply
532 (this, args);
533 }
534
535 }
536
537 };
538
539 /**
540 * This method will wrap the given
541 * tag around (and including)
542 * all elements
543 * between the startNode and
544 * endNode and try to maintain
545 * simple and valid
546 * HTML. The tag should be an "
547 * inline"-element, for "block"
548 * elements use
549 * {block}. Both methods have a
550 * different behaviour when
551 * generating markup.
552
553 * @param {String} tag
554 * @param {Node} startNode
555 * @param {Node} endNode
556 * @param {...*} [params]
557 * @returns {Element[]} - The
558 * elements created by the
559 * formatting function
560
561 selected area
562 if (enclosingTag = typeRange.
563 elementEnclosingStartAndEnd
564 (tag)) {
565 return this.removeInline(
566 enclosingTag, typeRange);
567 }
568 // Otherwise add formatting to
569 selected area
570 } else {
571 startNode = this._getStartNode
572 (tag, typeRange);
573 endNode = this._getEndNode(
574 tag, typeRange);
575 params = Array.prototype.
576 slice.call(arguments, 2);
577 args = [tag, startNode,
578 endNode].concat(params);
579 return this.insertInline.apply
580 (this, args);
581 }
582
583 }
584
585 };
586
587 /**
588 * This method will wrap the given
589 * tag around (and including)
590 * all elements
591 * between the startNode and
592 * endNode and try to maintain
593 * simple and valid
594 * HTML. The tag should be an "
595 * inline"-element, for "block"
596 * elements use
597 * {block}. Both methods have a
598 * different behaviour when
599 * generating markup.
600
601 * @param {String} tag
602 * @param {Node} startNode
603 * @param {Node} endNode
604 * @param {...*} [params]
605 * @returns {Element[]} - The
606 * elements created by the
607 * formatting function
608
609 selected area
610 if (enclosingTag = typeRange.
611 elementEnclosingStartAndEnd
612 (tag)) {
613 return this.removeInline(
614 enclosingTag, typeRange);
615 }
616 // Otherwise add formatting to
617 selected area
618 } else {
619 startNode = this._getStartNode
620 (tag, typeRange);
621 endNode = this._getEndNode(
622 tag, typeRange);
623 params = Array.prototype.
624 slice.call(arguments, 2);
625 args = [tag, startNode,
626 endNode].concat(params);
627 return this.insertInline.apply
628 (this, args);
629 }
630
631 }
632
633 };
634
635 /**
636 * This method will wrap the given
637 * tag around (and including)
638 * all elements
639 * between the startNode and
640 * endNode and try to maintain
641 * simple and valid
642 * HTML. The tag should be an "
643 * inline"-element, for "block"
644 * elements use
645 * {block}. Both methods have a
646 * different behaviour when
647 * generating markup.
648
649 * @param {String} tag
650 * @param {Node} startNode
651 * @param {Node} endNode
652 * @param {...*} [params]
653 * @returns {Element[]} - The
654 * elements created by the
655 * formatting function
656
657 selected area
658 if (enclosingTag = typeRange.
659 elementEnclosingStartAndEnd
660 (tag)) {
661 return this.removeInline(
662 enclosingTag, typeRange);
663 }
664 // Otherwise add formatting to
665 selected area
666 } else {
667 startNode = this._getStartNode
668 (tag, typeRange);
669 endNode = this._getEndNode(
670 tag, typeRange);
671 params = Array.prototype.
672 slice.call(arguments, 2);
673 args = [tag, startNode,
674 endNode].concat(params);
675 return this.insertInline.apply
676 (this, args);
677 }
678
679 }
680
681 };
682
683 /**
684 * This method will wrap the given
685 * tag around (and including)
686 * all elements
687 * between the startNode and
688 * endNode and try to maintain
689 * simple and valid
690 * HTML. The tag should be an "
691 * inline"-element, for "block"
692 * elements use
693 * {block}. Both methods have a
694 * different behaviour when
695 * generating markup.
696
697 * @param {String} tag
698 * @param {Node} startNode
699 * @param {Node} endNode
700 * @param {...*} [params]
701 * @returns {Element[]} - The
702 * elements created by the
703 * formatting function
704
705 selected area
706 if (enclosingTag = typeRange.
707 elementEnclosingStartAndEnd
708 (tag)) {
709 return this.removeInline(
710 enclosingTag, typeRange);
711 }
712 // Otherwise add formatting to
713 selected area
714 } else {
715 startNode = this._getStartNode
716 (tag, typeRange);
717 endNode = this._getEndNode(
718 tag, typeRange);
719 params = Array.prototype.
720 slice.call(arguments, 2);
721 args = [tag, startNode,
722 endNode].concat(params);
723 return this.insertInline.apply
724 (this, args);
725 }
726
727 }
728
729 };
730
731 /**
732 * This method will wrap the given
733 * tag around (and including)
734 * all elements
735 * between the startNode and
736 * endNode and try to maintain
737 * simple and valid
738 * HTML. The tag should be an "
739 * inline"-element, for "block"
740 * elements use
741 * {block}. Both methods have a
742 * different behaviour when
743 * generating markup.
744
745 * @param {String} tag
746 * @param {Node} startNode
747 * @param {Node} endNode
748 * @param {...*} [params]
749 * @returns {Element[]} - The
750 * elements created by the
751 * formatting function
752
753 selected area
754 if (enclosingTag = typeRange.
755 elementEnclosingStartAndEnd
756 (tag)) {
757 return this.removeInline(
758 enclosingTag, typeRange);
759 }
760 // Otherwise add formatting to
761 selected area
762 } else {
763 startNode = this._getStartNode
764 (tag, typeRange);
765 endNode = this._getEndNode(
766 tag, typeRange);
767 params = Array.prototype.
768 slice.call(arguments, 2);
769 args = [tag, startNode,
770 endNode].concat(params);
771 return this.insertInline.apply
772 (this, args);
773 }
774
775 }
776
777 };
778
779 /**
780 * This method will wrap the given
781 * tag around (and including)
782 * all elements
783 * between the startNode and
784 * endNode and try to maintain
785 * simple and valid
786 * HTML. The tag should be an "
787 * inline"-element, for "block"
788 * elements use
789 * {block}. Both methods have a
790 * different behaviour when
791 * generating markup.
792
793 * @param {String} tag
794 * @param {Node} startNode
795 * @param {Node} endNode
796 * @param {...*} [params]
797 * @returns {Element[]} - The
798 * elements created by the
799 * formatting function
800
801 selected area
802 if (enclosingTag = typeRange.
803 elementEnclosingStartAndEnd
804 (tag)) {
805 return this.removeInline(
806 enclosingTag, typeRange);
807 }
808 // Otherwise add formatting to
809 selected area
810 } else {
811 startNode = this._getStartNode
812 (tag, typeRange);
813 endNode = this._getEndNode(
814 tag, typeRange);
815 params = Array.prototype.
816 slice.call(arguments, 2);
817 args = [tag, startNode,
818 endNode].concat(params);
819 return this.insertInline.apply
820 (this, args);
821 }
822
823 }
824
825 };
826
827 /**
828 * This method will wrap the given
829 * tag around (and including)
830 * all elements
831 * between the startNode and
832 * endNode and try to maintain
833 * simple and valid
834 * HTML. The tag should be an "
835 * inline"-element, for "block"
836 * elements use
837 * {block}. Both methods have a
838 * different behaviour when
839 * generating markup.
840
841 * @param {String} tag
842 * @param {Node} startNode
843 * @param {Node} endNode
844 * @param {...*} [params]
845 * @returns {Element[]} - The
846 * elements created by the
847 * formatting function
848
849 selected area
850 if (enclosingTag = typeRange.
851 elementEnclosingStartAndEnd
852 (tag)) {
853 return this.removeInline(
854 enclosingTag, typeRange);
855 }
856 // Otherwise add formatting to
857 selected area
858 } else {
859 startNode = this._getStartNode
860 (tag, typeRange);
861 endNode = this._getEndNode(
862 tag, typeRange);
863 params = Array.prototype.
864 slice.call(arguments, 2);
865 args = [tag, startNode,
866 endNode].concat(params);
867 return this.insertInline.apply
868 (this, args);
869 }
870
871 }
872
873 };
874
875 /**
876 * This method will wrap the given
877 * tag around (and including)
878 * all elements
879 * between the startNode and
880 * endNode and try to maintain
881 * simple and valid
882 * HTML. The tag should be an "
883 * inline"-element, for "block"
884 * elements use
885 * {block}. Both methods have a
886 * different behaviour when
887 * generating markup.
888
889 * @param {String} tag
890 * @param {Node} startNode
891 * @param {Node} endNode
892 * @param {...*} [params]
893 * @returns {Element[]} - The
894 * elements created by the
895 * formatting function
896
897 selected area
898 if (enclosingTag = typeRange.
899 elementEnclosingStartAndEnd
900 (tag)) {
901 return this.removeInline(
902 enclosingTag, typeRange);
903 }
904 // Otherwise add formatting to
905 selected area
906 } else {
907 startNode = this._getStartNode
908 (tag, typeRange);
909 endNode = this._getEndNode(
910 tag, typeRange);
911 params = Array.prototype.
912 slice.call(arguments, 2);
913 args = [tag, startNode,
914 endNode].concat(params);
915 return this.insertInline.apply
916 (this, args);
917 }
918
919 }
920
921 };
922
923 /**
924 * This method will wrap the given
925 * tag around (and including)
926 * all elements
927 * between the startNode and
928 * endNode and try to maintain
929 * simple and valid
930 * HTML. The tag should be an "
931 * inline"-element, for "block"
932 * elements use
933 * {block}. Both methods have a
934 * different behaviour when
935 * generating markup.
936
937 * @param {String} tag
938 * @param {Node} startNode
939 * @param {Node} endNode
940 * @param {...*} [params]
941 * @returns {Element[]} - The
942 * elements created by the
943 * formatting function
944
945 selected area
946 if (enclosingTag = typeRange.
947 elementEnclosingStartAndEnd
948 (tag)) {
949 return this.removeInline(
950 enclosingTag, typeRange);
951 }
952 // Otherwise add formatting to
953 selected area
954 } else {
955 startNode = this._getStartNode
956 (tag, typeRange);
957 endNode = this._getEndNode(
958 tag, typeRange);
959 params = Array.prototype.
960 slice.call(arguments, 2);
961 args = [tag, startNode,
962 endNode].concat(params);
963 return this.insertInline.apply
964 (this, args);
965 }
966
967 }
968
969 };
970
971 /**
972 * This method will wrap the given
973 * tag around (and including)
974 * all elements
975 * between the startNode and
976 * endNode and try to maintain
977 * simple and valid
978 * HTML. The tag should be an "
979 * inline"-element, for "block"
980 * elements use
981 * {block}. Both methods have a
982 * different behaviour when
983 * generating markup.
984
985 * @param {String} tag
986 * @param {Node} startNode
987 * @param {Node} endNode
988 * @param {...*} [params]
989 * @returns {Element[]} - The
990 * elements created by the
991 * formatting function
992
993 selected area
994 if (enclosingTag = typeRange.
995 elementEnclosingStartAndEnd
996 (tag)) {
997 return this.removeInline(
998 enclosingTag, typeRange);
999 }
1000 // Otherwise add formatting to
1001 selected area
1002 } else {
1003 startNode = this._getStartNode
1004 (tag, typeRange);
1005 endNode = this._getEndNode(
1006 tag, typeRange);
1007 params = Array.prototype.
1008 slice.call(arguments, 2);
1009 args = [tag, startNode,
1010 endNode].concat(params);
1011 return this.insertInline.apply
1012 (this, args);
1013 }
1014
1015 }
1016
1017 };
1018
1019 /**
1020 * This method will wrap the given
1021 * tag around (and including)
1022 * all elements
1023 * between the startNode and
1024 * endNode and try to maintain
1025 * simple and valid
1026 * HTML. The tag should be an "
1027 * inline"-element, for "block"
1028 * elements use
1029 * {block}. Both methods have a
1030 * different behaviour when
1031 * generating markup.
1032
1033 * @param {String} tag
1034 * @param {Node} startNode
1035 * @param {Node} endNode
1036 * @param {...*} [params]
1037 * @returns {Element[]} - The
1038 * elements created by the
1039 * formatting function
1040
1041 selected area
1042 if (enclosingTag = typeRange.
1043 elementEnclosingStartAndEnd
1044 (tag)) {
1045 return this.removeInline(
1046 enclosingTag, typeRange);
1047 }
1048 // Otherwise add formatting to
1049 selected area
1050 } else {
1051 startNode = this._getStartNode
1052 (tag, typeRange);
1053 endNode = this._getEndNode(
1054 tag, typeRange);
1055 params = Array.prototype.
1056 slice.call(arguments, 2);
1057 args = [tag, startNode,
1058 endNode].concat(params);
1059 return this.insertInline.apply
1060 (this, args);
1061 }
1062
1063 }
1064
1065 };
1066
1067 /**
1068 * This method will wrap the given
1069 * tag around (and including)
1070 * all elements
1071 * between the startNode and
1072 * endNode and try to maintain
1073 * simple and valid
1074 * HTML. The tag should be an "
1075 * inline"-element, for "block"
1076 * elements use
1077 * {block}. Both methods have a
1078 * different behaviour when
1079 * generating markup.
1080
1081 * @param {String} tag
1082 * @param {Node} startNode
1083 * @param {Node} endNode
1084 * @param {...*} [params]
1085 * @returns {Element[]} - The
1086 * elements created by the
1087 * formatting function
1088
1089 selected area
1090 if (enclosingTag = typeRange.
1091 elementEnclosingStartAndEnd
1092 (tag)) {
1093 return this.removeInline(
1094 enclosingTag, typeRange);
1095 }
1096 // Otherwise add formatting to
1097 selected area
1098 } else {
1099 startNode = this._getStartNode
1100 (tag, typeRange);
1101 endNode = this._getEndNode(
1102 tag, typeRange);
1103 params = Array.prototype.
1104 slice.call(arguments, 2);
1105 args = [tag, startNode,
1106 endNode].concat(params);
1107 return this.insertInline.apply
1108 (this, args);
1109 }
1110
1111 }
1112
1113 };
1114
1115 /**
1116 * This method will wrap the given
1117 * tag around (and including)
1118 * all elements
1119 * between the startNode and
1120 * endNode and try to maintain
1121 * simple and valid
1122 * HTML. The tag should be an "
1123 * inline"-element, for "block"
1124 * elements use
1125 * {block}. Both methods have a
1126 * different behaviour when
1127 * generating markup.
1128
1129 * @param {String} tag
1130 * @param {Node} startNode
1131 * @param {Node} endNode
1132 * @param {...*} [params]
1133 * @returns {Element[]} - The
1134 * elements created by the
1135 * formatting function
1136
1137 selected area
1138 if (enclosingTag = typeRange.
1139 elementEnclosingStartAndEnd
1140 (tag)) {
1141 return this.removeInline(
1142 enclosingTag, typeRange);
1143 }
1144 // Otherwise add formatting to
1145 selected area
1146 } else {
1147 startNode = this._getStartNode
1148 (tag, typeRange);
1149 endNode = this._getEndNode(
1150 tag, typeRange);
1151 params = Array.prototype.
1152 slice.call(arguments, 2);
1153 args = [tag, startNode,
1154 endNode].concat(params);
1155 return this.insertInline.apply
1156 (this, args);
1157 }
1158
1159 }
1160
1161 };
1162
1163 /**
1164 * This method will wrap the given
1165 * tag around (and including)
1166 * all elements
1167 * between the startNode and
1168 * endNode and try to maintain
1169 * simple and valid
1170 * HTML. The tag should be an "
1171 * inline"-element, for "block"
1172 * elements use
1173 * {block}. Both methods have a
1174 * different behaviour when
1175 * generating markup.
1176
1177 * @param {String} tag
1178 * @param {Node} startNode
1179 * @param {Node} endNode
1180 * @param {...*} [params]
1181 * @returns {Element[]} - The
1182 * elements created by the
1183 * formatting function
1184
1185 selected area
1186 if (enclosingTag = typeRange.
1187 elementEnclosingStartAndEnd
1188 (tag)) {
1189 return this.removeInline(
1190 enclosingTag, typeRange);
1191 }
1192 // Otherwise add formatting to
1193 selected area
1194 } else {
1195 startNode = this._getStartNode
1196 (tag, typeRange);
1197 endNode = this._getEndNode(
1198 tag, typeRange);
1199 params = Array.prototype.
1200 slice.call(arguments, 2);
1201 args = [tag, startNode,
1202 endNode].concat(params);
1203 return this.insertInline.apply
1204 (this, args);
1205 }
1206
1207 }
1208
1209 };
1210
1211 /**
1212 * This method will wrap the given
1213 * tag around (and including)
1214 * all elements
1215 * between the startNode and
1216 * endNode and try to maintain
1217 * simple and valid
1218 * HTML. The tag should be an "
1219 * inline"-element, for "block"
1220 * elements use
1221 * {block}. Both methods have a
1222 * different behaviour when
1223 * generating markup.
1224
1225 * @param {String} tag
1226 * @param {Node} startNode
1227 * @param {Node} endNode
1228 * @param {...*} [params]
1229 * @returns {Element[]} - The
1230 * elements created by the
1231 * formatting function
1232
1233 selected area
1234 if (enclosingTag = typeRange.
1235 elementEnclosingStartAndEnd
1236 (tag)) {
1237 return this.removeInline(
1238 enclosingTag, typeRange);
1239 }
1240 // Otherwise add formatting to
1241 selected area
1242 } else {
1243 startNode = this._getStartNode
1244 (tag, typeRange);
1245 endNode = this._getEndNode(
1246 tag, typeRange);
1247 params = Array.prototype.
1248 slice.call(arguments, 2);
1249 args = [tag, startNode,
1250 endNode].concat(params);
1251 return this.insertInline.apply
1252 (this, args);
1253 }
1254
1255 }
1256
1257 };
1258
1259 /**
1260 * This method will wrap the given
1261 * tag around (and including)
1262 * all elements
1263 * between the startNode and
1264 * endNode and try to maintain
1265 * simple and valid
1266 * HTML. The tag should be an "
1267 * inline"-element, for "block"
1268 * elements use
1269 * {block}. Both methods have a
1270 * different behaviour when
1271 * generating markup.
1272
1273 * @param {String} tag
1274 * @param {Node} startNode
1275 * @param {Node} endNode
1276 * @param {...*} [params]
1277 * @returns {Element[]} - The
1278 * elements created by the
1279 * formatting function
1280
1281 selected area
1282 if (enclosingTag = typeRange.
1283 elementEnclosingStartAndEnd
1284 (tag)) {
1285 return this.removeInline(
1286 enclosingTag, typeRange);
1287 }
1288 // Otherwise add formatting to
1289 selected area
1290 } else {
1291 startNode = this._getStartNode
1292 (tag, typeRange);
1293 endNode = this._getEndNode(
1294 tag, typeRange);
1295 params = Array.prototype.
1296 slice.call(arguments, 2);
1297 args = [tag, startNode,
1298 endNode].concat(params);
1299 return this.insertInline.apply
1300 (this, args);
1301 }
1302
1303 }
1304
1305 };
1306
1307 /**
1308 * This method will wrap the given
1309 * tag around (and including)
1310 * all elements
1311 * between the startNode and
1312 * endNode and try to maintain
1313 * simple and valid
1314 * HTML. The tag should be an "
1315 * inline"-element, for "block"
1316 * elements use
1317 * {block}. Both methods have a
1318 * different behaviour when
1319 * generating markup.
1320
1321 * @param {String} tag
1322 * @param {Node} startNode
1323 * @param {Node} endNode
1324 * @param {...*} [params]
1325 * @returns {Element[]} - The
1326 * elements created by the
1327 * formatting function
1328
1329 selected area
1330 if (enclosingTag = typeRange.
1331 elementEnclosingStartAndEnd
1332 (tag)) {
1333 return this.removeInline(
1334 enclosingTag, typeRange);
1335 }
1336 // Otherwise add formatting to
1337 selected area
1338 } else {
1339 startNode = this._getStartNode
1340 (tag, typeRange);
1341 endNode = this._getEndNode(
1342 tag, typeRange);
1343 params = Array.prototype.
1344 slice.call(arguments, 2);
1345 args = [tag, startNode,
1346 endNode].concat(params);
1347 return this.insertInline.apply
1348 (this, args);
1349 }
1350
1351 }
1352
1353 };
1354
1355 /**
1356 * This method will wrap the given
1357 * tag around (and including)
1358 * all elements
1359 * between the startNode and
1360 * endNode and try to maintain
1361 * simple and valid
1362 * HTML. The tag should be an "
1363 * inline"-element, for "block"
1364 * elements use
1365 * {block}. Both methods have a
1366 * different behaviour when
1367 * generating markup.
1368
1369 * @param {String} tag
1370 * @param {Node} startNode
1371 * @param {Node} endNode
1372 * @param {...*} [params]
1373 * @returns {Element[]} - The
1374 * elements created by the
1375 * formatting function
1376
1377 selected area
1378 if (enclosingTag = typeRange.
1379 elementEnclosingStartAndEnd
1380 (tag)) {
1381 return this.removeInline(
1382 enclosingTag, typeRange);
1383 }
1384 // Otherwise add formatting to
1385 selected area
1386 } else {
1387 startNode = this._getStartNode
1388 (tag, typeRange);
1389 endNode = this._getEndNode(
1390 tag, typeRange);
1391 params = Array.prototype.
1392 slice.call(arguments, 2);
1393 args = [tag, startNode,
1394 endNode].concat(params);
```

```

116 */
117 this.insertInline = function (tag, 145
    {
118
119 // Required variables
120 var currentNode = startNode, 146
121 createdNodes = [], 147
122 nodesToWrap = [], 148
123 nextNode;
124
125 // Collect the startNode and all
    its siblings until we
126 // found the endNode or a node
    containing it
127 while (currentNode && !
    currentNode.contains(
        endNode)) {
128     nodesToWrap.push(currentNode);
129     currentNode = currentNode.
        nextSibling;
130 }
131
132 // If the node where we stopped
133 // is the endNode, add it
134 // to our collection of nodes
135 if (currentNode === endNode) {
136     nodesToWrap.push(currentNode);
137 }
138
139 // If the node where we stopped
140 // contains the endNode,
141 // apply this algorithm on it
142 // recursively
143 if (currentNode && Type.
    DomUtilities.
        containsButIsnt(currentNode
            , endNode)) {
144     createdNodes.concat(this.
        insertInline(tag,
            currentNode.firstChild,
            endNode));
145 }
146
147 // If we did not find the
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
    endNode but there are no
    more
    // siblings, find the next node
    in the document flow and
    // apply this algorithm on it
    recursively
    if (currentNode === null) {
        nextNode = Type.DomWalker.next
            (startNode.parentNode.
                lastChild, this._type.
                    getRoot());
        createdNodes.concat(this.
            insertInline(tag,
                nextNode, endNode));
    }
    // Wrap the nodes we got so far
    in the provided tag
    createdNodes.push(Type.
        DomUtilities.wrap(tag,
            nodesToWrap));
    // Return all nodes that have
    been created
    return createdNodes;
};
/**
 * @param {Node} enclosingTag
 * @param {Type.Range} typeRange
 * @returns {Type.Formatter}
 */
this.removeInline = function (
    enclosingTag, typeRange) {
    var tagName = enclosingTag.
        tagName,
        tagPositions = Type.Range.
            fromElement(enclosingTag)
                .save(this._type.getRoot
                    ()),
        selPositions = typeRange.save(
            this._type.getRoot()),
    leftRange,
    rightRange;
    Type.DomUtilities.unwrap(
        enclosingTag);
    leftRange = Type.Range.
        fromPositions(this._type.
            getRoot(), tagPositions.
                start, selPositions.start);
    if (!leftRange.isCollapsed()) {
        this.inline(tagName, leftRange
            );
    }
    rightRange = Type.Range.
        fromPositions(this._type.
            getRoot(), selPositions.end
                , tagPositions.end);
    if (!rightRange.isCollapsed()) {
        this.inline(tagName,
            rightRange);
    }
    return this;
};
/**
 * @param cmd
 * @param typeRange
 * @param params
 * @returns {Type.Formatter}
 * @private
 */
this.block = function (cmd,
    typeRange, params) {
    return this.inline.apply(this,
        arguments);
};
/**
 *

```


Listing 14: formatter.js

```

1 'use strict';
2
3 var Type = require('./core');
4
5 /**
6  * An editor's caret. We cannot use
7  * the browser's native caret
8  * since we do not utilize
9  * native inputs (a textarea or an
10 * element that is set to
11 * contenteditable). We emulate
12 * a caret with a blinking div. This
13 * class manages that div and
14 * provides methods to
15 * position it.
16
17 *
18 *
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
102
```

```

29 this.callbacks = {};
30
31 this.__constrainingNode = options.
32   constrainingNode || document.
33   body;
34 this.caretEl = this.__createElement
35   (options.color);
36 //this.moveTo(this.
37   __constrainingNode);
38 //this._hide();
39
40 };
41
42 (function () {
43   /**
44    * The id attribute of the caret
45    * container element as created
46    * by
47    * _getElementContainer()
48    */
49   @type {string}
50   @private
51   */
52   this.__containerId = Type.Settings.
53     prefix + 'caret-container';
54
55   /**
56    * Moves the caret left by one
57    * character
58    */
59   @returns {Type.Caret}
60   */
61   this.moveLeft = function () {
62     if (this.offset <= this.
63       _visibleTextOffsets(this.
64         textNode).start) {
65       var prevTextNode = this.
66         _prevTextNode(this.
67           textNode);
68       if (prevTextNode !== null) this
69         .moveTo(prevTextNode,
70           this._visibleTextOffsets(
71             prevTextNode).end);
72     }
73   }
74
75   /**
76    * Moves the caret right by one
77    * character
78    */
79   @returns {Type.Caret}
80   */
81   this.moveRight = function () {
82     if (this.offset >= this.
83       _visibleTextOffsets(this.
84         textNode).end) {
85       var nextTextNode = this.
86         _nextTextNode(this.
87           textNode);
88       if (nextTextNode !== null) this
89         .moveTo(nextTextNode,
90           this._visibleTextOffsets(
91             nextTextNode).start);
92     }
93   }
94
95   /**
96    * Moves the caret up by one line.
97    * Tries to preserve horizontal
98    * position.
99    */
100   Moves the caret up by one line.
101   Tries to preserve horizontal
102   position.
103   */
104   this.moveUp = function () {
105     // TODO prevNode handling not nice
106     // TODO should only walk 1 line
107     // Internally, this will create a
108     // collapsed range at the caret
109     // 's offset and move
110     // it left, character by character
111     // and stop in the line above
112     // the caret when it's
113     // horizontally aligned with it.
114     // The caret will then be moved
115     // to that position.
116     @returns {Type.Caret}
117     */
118     this.moveUp = function () {
119       // Shorthand variables
120       var node = this.textNode,
121         offset = this.offset,
122         prevNode = node;
123
124       // Initial range and positions
125       var range = this._createRange(
126         node, offset),
127         rangePos = this.
128           _getPositionsFromRange(
129             range),
130         caretPos = this.
131           _getRectAtOffset(this.
132             offset),
133         lastRangeLeft;
134
135       // Move the range as described
136       // in the method's description
137       while( prevNode !== null // &&
138         offset > 0 &&
139         && (rangePos || (rangePos
140           .top == caretPos.top
141           || rangePos.left >
142             caretPos.left)) ) {
143         if (offset <= 0) {
144           prevNode = this.
145             _prevTextNode(node);
146           if (prevNode !== null) {
147             node = prevNode;
148             offset = prevNode.length;
149             // TODO Check auf !
150             // noetig wenn
151             _visibleTextOffsets
152               verwendet werden, da
153             //
154           }
155         }
156       }
157     }
158   }

```

```

114         unsichtbarer text nie
115         selektiert wird
116     }
117     } else {
118         offset--;
119     }
120     range.setStart(node, offset);
121     range.collapse(true);
122     lastRangeLeft = rangePos.left;
123     rangePos = this;
124     _getPositionsFromRange(
125         range);
126 }
127 // If the range moved up, check
128 // 2 characters above the
129 // caret to find a precise pos
130 .
131 if (rangePos.top < caretPos.top)
132 {
133     if (this._compareDeltaTo(
134         caretPos.left,
135         lastRangeLeft, rangePos.
136         left) == -1) {
137         offset += 1;
138     }
139     this.moveTo(node, offset);
140 }
141 // Chaining
142 return this;
143 };
144 /**
145  * Moves the caret down by one
146  * line.
147  * Tries to preserve horizontal
148  * position.
149  *
150  * Todo nextNode handling not nice
151  * Todo should only walk 1 line
152  * @returns {Type.Caret}
153  */
154 this.moveDown = function () {
155
156     // Shorthand variables
157     var node = this.textNode,
158         offset = this.offset,
159         nextNode = node;
160
161     // We are gonna create a range
162     // and move it through
163     // the text until it is
164     // positioned 1 line below
165     // the caret's position at
166     // around the same horizontal
167     // position
168     var range = this;
169     _createRange(node, offset),
170     rangePos = this;
171     _getPositionsFromRange(
172         range),
173     caretPos = this;
174     _getRectAtOffset(this,
175         offset),
176     visibleText = this;
177     _visibleTextOffsets(node)
178     ,
179     lastRangeRight;
180
181     // Move the range right letter
182     // by letter. The range will
183     // start
184     // in the same line and we keep
185     // moving it until it reaches
186     // the
187     // next line and stop moving
188     // when it has moved further
189     // right
190     // than the caret. That means
191     // the range will be one line
192     // below
193     // the caret and in about the
194     // same horizontal position.
195     while( nextNode !== null) {
196         offset < node.length &&
197         (!rangePos || (rangePos.
198             bottom == caretPos.bottom
199             || rangePos.right <
200                 caretPos.right))
201     }
202
203     // The text might have only one
204     // line, we check to see if
205     // the range
206     // has actually moved lower than
207     // the caret and then move
208     // the caret
209     // In any case we moved the
210     // offset too far by 1
211     // character so we
212     // need to subtract it
213     if (rangePos.bottom > caretPos.
214         bottom) {
215         if (this._compareDeltaTo(
216             caretPos.right,
217             lastRangeRight, rangePos.
218             right) == -1) {
219             offset -= 1;
220         }
221     }
222     // TODO gucken ob sich das
223     // noch irgendwie aufhängen
224     // kann wenn caret am ende des
225     // textes ist und rangePos
226     // nicht gesetzt ist
227     if (offset >= visibleText.end
228         /*node.length*/) {
229         nextNode = this;
230         _nextTextNode(node);
231     }
232     if (nextNode !== null) {
233         node = nextNode;
234         visibleText = this;
235         _visibleTextOffsets(
236             node);
237         offset = 0;
238     }
239     } else {
240         offset++;
241     }
242     range.setEnd(node, offset);
243     range.collapse(false);
244     lastRangeRight = rangePos.
245         right;
246     rangePos = this;
247     _getPositionsFromRange(
248         range);
249 }
250
251 // The text might have only one
252 // line, we check to see if
253 // the range
254 // has actually moved lower than
255 // the caret and then move
256 // the caret
257 // In any case we moved the
258 // offset too far by 1
259 // character so we
260 // need to subtract it
261 if (rangePos.bottom > caretPos.
262     bottom) {
263     if (this._compareDeltaTo(
264         caretPos.right,
265         lastRangeRight, rangePos.
266         right) == -1) {
267         offset -= 1;
268     }
269 }

```

```

193     }
194     this.moveTo(node, offset);
195 }
196 // Chaining
197 return this;
198 };
199 /**
200  * Moves the charat by the number
201  * of chars passed to as
202  * numChars
203  * @param {number} numChars - The
204  * number of chars the caret
205  * should be moved by.
206  * Accepts negative values.
207  * @returns {*}
208  */
209 this.moveBy = function (numChars)
210 {
211     var offset = this.getOffset();
212     if (offset === null) return this
213     this.setOffset(Math.max(0, this.
214         getOffset() + numChars));
215     return this;
216 };
217 /**
218  * Places the caret in a text node
219  * at a given position
220  * @param {Node} node - The (text)
221  * {Node} in which the caret
222  * should be placed
223  * @param {number} |offset=0| -
224  * The character offset where
225  * the caret should be moved to
226  * @returns {Type.Caret}
227  */
228 this.moveTo = function (node,
229     offset) {
230     if (node.nodeType !== Node.
231         TEXT_NODE) {
232         node = Type.DomWalker.first(
233
234             node, 'text');
235     }
236     if (node === null) {
237         throw new Error('Node
238             parameter must be or
239             contain a text node');
240     }
241     if (node === this.textNode &&
242         offset === null) {
243         return this;
244     }
245     this.textNode = node;
246     this._setOffset(offset || 0);
247     return this;
248 };
249 /**
250  * Inserts a given {string} at the
251  * caret's current offset in
252  * the caret's
253  * current text node
254  * *
255  * Todo this method needs to go
256  * somewhere else
257  * @param {string} str - The {
258  * string} that will be be
259  * inserted
260  * @returns {Type.Caret}
261  */
262 this.insertText = function (str) {
263     this._callBacksFor('insertText',
264         str);
265     if (/^\n\r|$/ .test(str)) {
266         var newNode = this.textNode.
267             splitText(this.offset);
268         newNode.parentNode.
269             insertBefore(document.
270                 createElement('br'),
271                 newNode);
272         this.moveTo(newNode, 0);
273     }
274 }
275
276     return this;
277 } else {
278     var nodeText = this.textNode.
279         nodeValue;
280     if (this.offset > 0) {
281         nodeText = nodeText.substring(
282             0, this.offset)
283         + str
284         + nodeText.substring(this.
285             offset, nodeText.length
286             );
287     } else {
288         nodeText = str + nodeText;
289     }
290 }

```

```

286 splitText = nodeText.split(/(?:\
287   r\n|\\r|\\n)/g);
288
289 this.textNode.nodeValue =
290   splitText[0];
291
292 for(i=1; i<splitText.length; i
293   ++){
294   tmpNode = document.
295     createTextNode(splitText[
296       i]);
297   parentNode.insertBefore(
298     tmpNode, this.textNode.
299     nextSibling);
300   if(i < splitText.length - 1)
301     parentNode.insertBefore(
302       document.createElement
303         ('br'), this.textNode.
304         nextSibling);
305   }
306
307 this.moveTo(tmpNode, tmpNode.
308   length);
309
310 */
311 };
312
313 /** Removes one character left from
314   the current offset
315   * and moves the caret accordingly
316   */
317
318 * Todo this method needs to go
319   somewhere else
320
321 * @param {number} [numChars] -
322   Home many characters should
323   be removed
324   * from the caret's position.
325   A negative number will
326   remove
327   characters left from the
328   caret, a positive number
329   from the right.
330   * @returns {Type.Caret}
331
332 this.removeCharacter = function (
333   r\n|\\r|\\n)/g);
334
335 this.textNode.nodeValue =
336   splitText[0];
337
338 for(i=1; i<splitText.length; i
339   ++){
340   tmpNode = document.
341     createTextNode(splitText[
342       i]);
343   parentNode.insertBefore(
344     tmpNode, this.textNode.
345     nextSibling);
346   if(i < splitText.length - 1)
347     parentNode.insertBefore(
348       document.createElement
349         ('br'), this.textNode.
350         nextSibling);
351   }
352
353 this.moveTo(tmpNode, tmpNode.
354   length);
355
356 */
357 };
358
359 /** Removes one character left from
360   the current offset
361   * and moves the caret accordingly
362   */
363
364 * Todo this method needs to go
365   somewhere else
366
367 * @param {number} [numChars] -
368   Home many characters should
369   be removed
370   * from the caret's position.
371   A negative number will
372   remove
373   characters left from the
374   caret, a positive number
375   from the right.
376   * @returns {Type.Caret}
377
378 this.removeCharacter = function (
379   r\n|\\r|\\n)/g);
380
381 this.textNode.nodeValue =
382   splitText[0];
383
384 for(i=1; i<splitText.length; i
385   ++){
386   tmpNode = document.
387     createTextNode(splitText[
388       i]);
389   parentNode.insertBefore(
390     tmpNode, this.textNode.
391     nextSibling);
392   if(i < splitText.length - 1)
393     parentNode.insertBefore(
394       document.createElement
395         ('br'), this.textNode.
396         nextSibling);
397   }
398
399 this.moveTo(tmpNode, tmpNode.
400   length);
401
402 */
403 };
404
405 /** Removes one character left from
406   the current offset
407   * and moves the caret accordingly
408   */
409
410 * Todo this method needs to go
411   somewhere else
412
413 * @param {number} [numChars] -
414   Home many characters should
415   be removed
416   * from the caret's position.
417   A negative number will
418   remove
419   characters left from the
420   caret, a positive number
421   from the right.
422   * @returns {Type.Caret}
423
424 this.removeCharacter = function (
425   r\n|\\r|\\n)/g);
426
427 this.textNode.nodeValue =
428   splitText[0];
429
430 for(i=1; i<splitText.length; i
431   ++){
432   tmpNode = document.
433     createTextNode(splitText[
434       i]);
435   parentNode.insertBefore(
436     tmpNode, this.textNode.
437     nextSibling);
438   if(i < splitText.length - 1)
439     parentNode.insertBefore(
440       document.createElement
441         ('br'), this.textNode.
442         nextSibling);
443   }
444
445 this.moveTo(tmpNode, tmpNode.
446   length);
447
448 */
449 };
450
451 /** Removes one character left from
452   the current offset
453   * and moves the caret accordingly
454   */
455
456 * Todo this method needs to go
457   somewhere else
458
459 * @param {number} [numChars] -
460   Home many characters should
461   be removed
462   * from the caret's position.
463   A negative number will
464   remove
465   characters left from the
466   caret, a positive number
467   from the right.
468   * @returns {Type.Caret}
469
470 this.removeCharacter = function (
471   r\n|\\r|\\n)/g);
472
473 this.textNode.nodeValue =
474   splitText[0];
475
476 for(i=1; i<splitText.length; i
477   ++){
478   tmpNode = document.
479     createTextNode(splitText[
480       i]);
481   parentNode.insertBefore(
482     tmpNode, this.textNode.
483     nextSibling);
484   if(i < splitText.length - 1)
485     parentNode.insertBefore(
486       document.createElement
487         ('br'), this.textNode.
488         nextSibling);
489   }
490
491 this.moveTo(tmpNode, tmpNode.
492   length);
493
494 */
495 };
496
497 /** Removes one character left from
498   the current offset
499   * and moves the caret accordingly
500   */
501
502 * Todo this method needs to go
503   somewhere else
504
505 * @param {number} [numChars] -
506   Home many characters should
507   be removed
508   * from the caret's position.
509   A negative number will
510   remove
511   characters left from the
512   caret, a positive number
513   from the right.
514   * @returns {Type.Caret}
515
516 this.removeCharacter = function (
517   r\n|\\r|\\n)/g);
518
519 this.textNode.nodeValue =
520   splitText[0];
521
522 for(i=1; i<splitText.length; i
523   ++){
524   tmpNode = document.
525     createTextNode(splitText[
526       i]);
527   parentNode.insertBefore(
528     tmpNode, this.textNode.
529     nextSibling);
530   if(i < splitText.length - 1)
531     parentNode.insertBefore(
532       document.createElement
533         ('br'), this.textNode.
534         nextSibling);
535   }
536
537 this.moveTo(tmpNode, tmpNode.
538   length);
539
540 */
541 };
542
543 /** Removes one character left from
544   the current offset
545   * and moves the caret accordingly
546   */
547
548 * Todo this method needs to go
549   somewhere else
550
551 * @param {number} [numChars] -
552   Home many characters should
553   be removed
554   * from the caret's position.
555   A negative number will
556   remove
557   characters left from the
558   caret, a positive number
559   from the right.
560   * @returns {Type.Caret}
561
562 this.removeCharacter = function (
563   r\n|\\r|\\n)/g);
564
565 this.textNode.nodeValue =
566   splitText[0];
567
568 for(i=1; i<splitText.length; i
569   ++){
570   tmpNode = document.
571     createTextNode(splitText[
572       i]);
573   parentNode.insertBefore(
574     tmpNode, this.textNode.
575     nextSibling);
576   if(i < splitText.length - 1)
577     parentNode.insertBefore(
578       document.createElement
579         ('br'), this.textNode.
580         nextSibling);
581   }
582
583 this.moveTo(tmpNode, tmpNode.
584   length);
585
586 */
587 };
588
589 /** Removes one character left from
590   the current offset
591   * and moves the caret accordingly
592   */
593
594 * Todo this method needs to go
595   somewhere else
596
597 * @param {number} [numChars] -
598   Home many characters should
599   be removed
600   * from the caret's position.
601   A negative number will
602   remove
603   characters left from the
604   caret, a positive number
605   from the right.
606   * @returns {Type.Caret}
607
608 this.removeCharacter = function (
609   r\n|\\r|\\n)/g);
610
611 this.textNode.nodeValue =
612   splitText[0];
613
614 for(i=1; i<splitText.length; i
615   ++){
616   tmpNode = document.
617     createTextNode(splitText[
618       i]);
619   parentNode.insertBefore(
620     tmpNode, this.textNode.
621     nextSibling);
622   if(i < splitText.length - 1)
623     parentNode.insertBefore(
624       document.createElement
625         ('br'), this.textNode.
626         nextSibling);
627   }
628
629 this.moveTo(tmpNode, tmpNode.
630   length);
631
632 */
633 };
634
635 /** Removes one character left from
636   the current offset
637   * and moves the caret accordingly
638   */
639
640 * Todo this method needs to go
641   somewhere else
642
643 * @param {number} [numChars] -
644   Home many characters should
645   be removed
646   * from the caret's position.
647   A negative number will
648   remove
649   characters left from the
650   caret, a positive number
651   from the right.
652   * @returns {Type.Caret}
653
654 this.removeCharacter = function (
655   r\n|\\r|\\n)/g);
656
657 this.textNode.nodeValue =
658   splitText[0];
659
660 for(i=1; i<splitText.length; i
661   ++){
662   tmpNode = document.
663     createTextNode(splitText[
664       i]);
665   parentNode.insertBefore(
666     tmpNode, this.textNode.
667     nextSibling);
668   if(i < splitText.length - 1)
669     parentNode.insertBefore(
670       document.createElement
671         ('br'), this.textNode.
672         nextSibling);
673   }
674
675 this.moveTo(tmpNode, tmpNode.
676   length);
677
678 */
679 };
680
681 /** Removes one character left from
682   the current offset
683   * and moves the caret accordingly
684   */
685
686 * Todo this method needs to go
687   somewhere else
688
689 * @param {number} [numChars] -
690   Home many characters should
691   be removed
692   * from the caret's position.
693   A negative number will
694   remove
695   characters left from the
696   caret, a positive number
697   from the right.
698   * @returns {Type.Caret}
699
700 this.removeCharacter = function (
701   r\n|\\r|\\n)/g);
702
703 this.textNode.nodeValue =
704   splitText[0];
705
706 for(i=1; i<splitText.length; i
707   ++){
708   tmpNode = document.
709     createTextNode(splitText[
710       i]);
711   parentNode.insertBefore(
712     tmpNode, this.textNode.
713     nextSibling);
714   if(i < splitText.length - 1)
715     parentNode.insertBefore(
716       document.createElement
717         ('br'), this.textNode.
718         nextSibling);
719   }
720
721 this.moveTo(tmpNode, tmpNode.
722   length);
723
724 */
725 };
726
727 /** Removes one character left from
728   the current offset
729   * and moves the caret accordingly
730   */
731
732 * Todo this method needs to go
733   somewhere else
734
735 * @param {number} [numChars] -
736   Home many characters should
737   be removed
738   * from the caret's position.
739   A negative number will
740   remove
741   characters left from the
742   caret, a positive number
743   from the right.
744   * @returns {Type.Caret}
745
746 this.removeCharacter = function (
747   r\n|\\r|\\n)/g);
748
749 this.textNode.nodeValue =
750   splitText[0];
751
752 for(i=1; i<splitText.length; i
753   ++){
754   tmpNode = document.
755     createTextNode(splitText[
756       i]);
757   parentNode.insertBefore(
758     tmpNode, this.textNode.
759     next
```

```

372 };
373
374 /**
375  * todo unify with moveTo API
376  * @param offset
377  * @returns {*}
378  */
379 this.setOffset = function (offset)
380 {
381     var t = Type.TextWalker.nodeAt(
382         this._constrainingNode,
383         offset);
384     this.moveTo(t.node, t.offset);
385     return this;
386 };
387
388 /**
389  * Returns the offset of the caret
390  * relative to its current
391  * text node
392  * todo Use this method on every
393  * public access to this
394  * variable
395  * todo make offset private
396  * @returns {number|null}
397  */
398 this.getNodeOffset = function () {
399     return this.offset;
400 };
401
402 /**
403  * Getter for this instance's text
404  * node
405  * todo Use this method on every
406  * public access to this
407  * variable
408  * todo make textNode private
409  * @returns {Node|null}
410  */
411 this.getNode = function () {
412     return this.textNode;
413 };
414
415 /**
416  * Sets the offset and displays
417  */
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

477 this._removeClass(this.caretEl,
478 'hide');
479 this._addClass(this.caretEl, '
500 blink');
480 return this;
481 };
482 /**
483  * Hides the caret
484  * @returns {Type.Caret}
485  */
486 this._hide = function () {
487 this._removeClass(this.caretEl,
488 'blink');
489 this._addClass(this.caretEl, '
510 hide');
490 return this;
491 };
492 /**
493  * Resets the blink animation by
494  * recreating the caret div
495  * element
496  * Todo Maybe find a better way to
497  * reset the blink animation,
498  * DOM = slow
499  */
500 this._resetBlink = function () {
501 var newCaret = this.caretEl.
502 cloneNode(true);
503 this.caretEl.parentNode.
504 replaceChild(newCaret, this
505 .caretEl);
506 this.caretEl = newCaret;
507 return this;
508 };
509 /**
510  * Todo Maybe make a magic
511  * function that calls
512  */
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

567 * @returns {*}
568 * @private
569 */
570 this._prevTextNode = function (el,
571   returnMe) {
572   var parent = el.parentNode;
573   if (returnMe === true && this._
574     isTextNodeWithContents(el)
575     ) {
576     return el;
577   }
578   if (el.childNodes.length) {
579     return this._prevTextNode(el.
580       childNodes[el.childNodes.
581         length - 1], true);
582   }
583   if (el.previousSibling !== null) {
584     return this._prevTextNode(el.
585       previousSibling, true);
586   }
587   while (parent !== this.
588     if (parent.previousSibling !==
589       null) {
590     return this._prevTextNode(
591       parent.previousSibling
592       , true);
593     parent = parent.parentNode;
594   }
595   return null;
596 };
597 /**
598 * Todo: code duplication in
599   browser.js, there should be
600   a dom util module
601
602   598 * @param node
603   599 * @returns {boolean}
604   600 * @private
605   601 */
602   this._isTextNodeWithContents =
603     function (node) {
604     return node.nodeType === 3 &&
605       /\^\t\n\r |\/.test(node.
606         textContent);
607   };
608   /**
609   609 * Finds the whitespace at the
610   610 beginning and the end of a
611   611 text node and
612   612 returns their lengths
613   613 * @param textNode
614   614 * @returns {{start: number, end:
615     number}}
616   615 * @private
617   616 */
617   this._visibleTextOffsets =
618     function (textNode) {
619     var startWhitespace = textNode.
620       nodeValue.match(/\^\t\n\r
621       |+/g) || [''];
622     var endWhitespace = textNode.
623       nodeValue.match(/\t\n\r |+/
624       $/g) || [''];
625     return {
626       start : startWhitespace[0].
627         length,
628       end : textNode.nodeValue.
629         length - endWhitespace
630         [0].length
631     };
632   };
633   /**
634   633 Utility method to add a class
635   634 to an element
636   635 Todo There should be a separate
637   636 utility module for stuff
638   637 like this - yes
639   638
640   639   640   641   642   643   644   645
646   646   647   648   649   650   651   652   653   654
641   642   643   644   645   646   647   648   649   650   651   652   653   654
647   648   649   650   651   652   653   654
643   644   645   646   647   648   649   650   651   652   653   654
649   650   651   652   653   654
645   646   647   648   649   650   651   652   653   654
651   652   653   654
653   654
654   655   656   657   658   659   660   661   662   663   664   665   666   667   668   669   670   671   672   673   674   675   676   677   678   679   680   681   682   683   684   685   686   687   688   689   690   691   692   693   694   695   696   697   698   699   700   701   702   703   704   705   706   707   708   709   710   711   712   713   714   715   716   717   718   719   720   721   722   723   724   725   726   727   728   729   730   731   732   733   734   735   736   737   738   739   740   741   742   743   744   745   746   747   748   749   750   751   752   753   754   755   756   757   758   759   760   761   762   763   764   765   766   767   768   769   770   771   772   773   774   775   776   777   778   779   780   781   782   783   784   785   786   787   788   789   790   791   792   793   794   795   796   797   798   799   800   801   802   803   804   805   806   807   808   809   810   811   812   813   814   815   816   817   818   819   820   821   822   823   824   825   826   827   828   829   830   831   832   833   834   835   836   837   838   839   840   841   842   843   844   845   846   847   848   849   850   851   852   853   854   855   856   857   858   859   860   861   862   863   864   865   866   867   868   869   870   871   872   873   874   875   876   877   878   879   880   881   882   883   884   885   886   887   888   889   890   891   892   893   894   895   896   897   898   899   900   901   902   903   904   905   906   907   908   909   910   911   912   913   914   915   916   917   918   919   920   921   922   923   924   925   926   927   928   929   930   931   932   933   934   935   936   937   938   939   940   941   942   943   944   945   946   947   948   949   950   951   952   953   954   955   956   957   958   959   960   961   962   963   964   965   966   967   968   969   970   971   972   973   974   975   976   977   978   979   980   981   982   983   984   985   986   987   988   989   990   991   992   993   994   995   996   997   998   999

```



```

655         .join('|') + '(\|b|$)', '
656         el.className = el.className.
657         replace(regex, '');
658     }
659     return this;
660 };
661
662 /**
663  * Calculates The delta between a
664  * given the pivot (a {number})
665  * and a as
666  * returns -1 if a is closer to
667  * pivot,
668  * 1 if b is closer to pivot and 0
669  * if both numbers are equally
670  * close.
671
672  *
673  * @param {number} pivot - A
674  * number to which a and b
675  * will be compared to
676  * @param {number} a - An
677  * arbitrary number
678  * @param {number} b - An
679  * arbitrary number
680  * @returns {number}
681  * @private
682
683  this._compareDeltaTo = function (
684     pivot, a, b) {
685     var deltaA = Math.abs(pivot - a)
686     , deltaB = Math.abs(pivot - b)
687     ;
688     if (deltaA == deltaB) return 0;
689     return deltaA < deltaB ? -1 : 1;
690 };
691
692 /**
693  * Returns a {ClientRect} with the
694  * boundaries enclosing a
695  * character at a
696  * given offset in a text node
697
698  *
699  * @param {Node} [node=this.
700  textNode] - The text node
701  which containing the
702  character we which to fetch
703  the boundaries of.
704  * @param {number} offset - The
705  offset of the character we
706  which to fetch
707  the boundaries of
708  * @returns {{top: number, right:
709  number, bottom: number, left
710  : number}}
711  * @private
712  */
713 this._getRectAtOffset = function (
714     node, offset) {
715     if (typeof node == "number") {
716         offset = node;
717         node = this.textNode;
718     }
719     return this.
720         _getPositionsFromRange(this
721         ._createRange(node, offset)
722         );
723 };
724
725 /**
726  * Returns the positions from a {
727  ClientRect} relative to the
728  scroll
729  position
730
731  *
732  * @param {Range} range The {Range
733  } that should be measured
734  * @returns {{top: number, right:
735  number, bottom: number, left
736  : number}}
737  * @private
738  */
739 this._getPositionsFromRange =
740     function (range) {
741     var scroll = this.
742         _getScrollPosition();
743     var rect = range.getClientRects
744     () [0];
745
746     if (!rect) {
747         return false;
748     }
749     return {
750         top : rect.top + scroll.top,
751         right : rect.right + scroll.
752         left,
753         bottom : rect.bottom + scroll.
754         top,
755         left : rect.left + scroll.
756         left
757     };
758 };
759
760 /**
761  * Return's the window's
762  horizontal an vertical
763  scroll positions
764
765  * @returns {{top: (number), left:
766  (number)}}
767  * @private
768  */
769 this._getScrollPosition = function
770     () {
771     return {
772         top : window.pageYOffset ||
773         document.documentElement.
774         scrollTop,
775         left : window.pageXOffset ||
776         document.documentElement.
777         scrollLeft
778     };
779 };
780
781 /**
782  * Creates a {Range} and returns
783  it
784
785  * @param {Node} startNode - The
786  node in which the created
787  range should begin
788  * @param {number} start - The
789  offset at which the range
790  should start
791
792  *
793  */

```

```

739 * @param {number} [end=start] -
740   The offset at which the
741   range should end
742 *   Optional. Defaults to the
743   start offset.
744 * @param {Node} [endNode=node] -
745   The node in which the
746   created range should end.
747 *   Optional. Defaults to the
748   start node.
749 * @returns {Range}
750 * @private
751 this._createRange = function(
752   startNode, start, end,
753   endNode) {
754   var range = window.document.
755     createRange();
756   range.setEnd(endNode, ||
757     startNode, end || start);
758   range.setStart(startNode, start)
759   return range;
760 };
761 /**
762 * Creates a div (the visual
763   representation of the caret)
764
765   and returns it.
766
767   * @returns {HTMLElement}
768   * @private
769   */
770 this._createElement = function (
771   color) {
772   var container = this.
773     _getElementContainer(),
774     el = window.document.
775       createElement('div');
776   el.className = Type.Settings.
777     prefix + 'caret' + color;
778   container.appendChild(el);
779   return el;
780 };
781 /**
782 * All div representations of
783   carets will be appended to a
784   single
785   container. This method returns
786   this container and creates
787   it
788   * if it has not been created yet.
789   * Todo use container from
790
791   dom_utilites
792   * @returns {HTMLElement}
793   * @private
794   */
795 this._getElementContainer =
796   function () {
797     var container = window.document.
798       getElementById(this.
799         _containerId);
800     if (container === null) {
801       container = window.document.
802         createElement('div');
803       container.setAttribute('id',
804         this._containerId);
805       window.document.body.
806         appendChild(container);
807     }
808     return container;
809   }
810   }.call(Type.Caret.prototype);
811 module.exports = Type.Caret;
812
813 Listing 15: caret.js
814
815 this._setValues(x1, y1, x2, y2);
816 this._anchor = {x: x1, y: y1};
817 };
818 (function () {
819   /**
820    * Will set the position and
821    dimension values and update
822    the div styles
823    * @param {number|string} [x1] -
824    Horizontal position of the
825    overlay
826    * or either one of the
827
828   'use strict';
829   var Type = require('./core');
830
831   /**
832    * todo internal differentiation /
833    abstraction of x and y *and
834    scroll positions* for easier
835    redrawing
836
837   * @param {number} [x1] - Horizontal
838   position of the overlay
839   * @param {number} [y1] - Vertical
840   position of the overlay
841
842   * @param {number} [x2] - x2 of the
843   overlay
844   * @param {number} [y2] - y2 of the
845   overlay
846   * @param {boolean} [show] - Set to
847   false if you do not wish
848   for the element to be shown.
849   * Defaults to true
850   * @constructor
851   Type.SelectionOverlay = function (x1
852     , y1, x2, y2, show) {
853     if (show !== false) {
854       this.show(x1, y1, x2, y2);
855     }
856
857   }

```

```

33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63

strings 'left', 'right' or '
line', which
* will span the overlay to
the left side of the line of
the
* textNode, the right side or
the entire line
* @param {number} [y1] - Vertical
position of the overlay
* @param {number} [x2] - x2 of
the overlay
* @param {number} [y2] - y2 of
the overlay
* @returns {Type.SelectionOverlay
} - This instance
*/
this.set = function (x1, y1, x2,
y2) {
    if (x1 === 'left') {
        x1 = this._textleft();
        x2 = null;
    }
    if (x1 === 'right') {
        x1 = null;
        x2 = this._textRight();
    }
    if (x1 === 'line') {
        x1 = this._textleft();
        x2 = this._textRight();
    }
    x1 = x1 === undefined ? null :
x1;
y1 = y1 === undefined ? null :
y1;
x2 = x2 === undefined ? null :
x2;
y2 = y2 === undefined ? null :
y2;
    this._draw(x1, y1, x2, y2);
    this._setValues(x1, y1, x2, y2);
101

102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134

return this;
};
/**
 * @param x1
 * @param y1
 * @param x2
 * @param y2
 * @returns {Type.SelectionOverlay
} - This instance
*/
this.show = function (x1, y1, x2,
y2) {
    this._el = this._createElement();
    this._draw(x1, y1, x2, y2);
    return this;
};
/**
 * @returns {Type.SelectionOverlay
} - This instance
*/
this.hide = function () {
    Type.DomUtilities.removeElement(
this._el);
    this._el = null;
    return this;
};
/**
 * @param {number|string} x
 * @param {number} [y]
 * @returns {Type.SelectionOverlay
} - This instance
*/
this.anchor = function (x, y) {
    if (x === 'left') {
        x = this._textleft();
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134

y = null;
    if (x === 'right') {
        x = this._textRight();
        y = null;
    }
    if (x !== null && x !==
undefined) {
        this._anchor.x = x;
    }
    if (x !== null && y !==
undefined) {
        this._anchor.y = y;
    }
    return this;
};
/**
 * Sets the horizontal start or
end of this overlay
depending
 * whether the value given is left
or right of the anchor.
 * Will also set the other end to
the anchor's position.
 * @param {number} x - The
horizontal position
 * @returns {Type.SelectionOverlay
} - This instance
*/
this.setXFromAnchor = function (
x) {
    if (x === null || x ===
undefined) {
        this.set(this._anchor.x, null,
this._anchor.x, null);
    }
    else {
        if (x < this._anchor.x) this.

```

```

135         set(x, null, this._anchor
136             .x, null);
137         if (x > this._anchor.x) this.
138             set(this._anchor.x, null,
139                 x, null);
140     }
141     return this;
142 };*/
143 /**
144  * Returns whether or not this
145  * overlay is actually visible
146  */
147 * @returns {boolean}
148 */
149 this.visible = function () {
150     return !(this.x1 === this.x2 ||
151             this.y1 === this.y2);
152 };
153 /**
154  * Removes the overlay div and
155  * resets all position and
156  * dimension values
157  */
158 * @returns {Type.SelectionOverlay}
159 } - This instance
160 */
161 this.remove = function () {
162     if (this._el) {
163         Type.DomUtilities.
164             removeElement(this._el);
165     }
166     this._el = null;
167     this.x1 = null;
168     this.y1 = null;
169     this.x2 = null;
170     this.y2 = null;
171     this._anchor = null;
172     return this;
173 };
174 /**
175  * Sets all dimension and position
176  */
177 this._draw = function (x1, y1, x2, y2) {
178     // values to the given
179     // values unless null is given
180     * @param {number} [x1] -
181     Horizontal position of the
182     overlay
183     * @param {number} [y1] - Vertical
184     position of the overlay
185     * @param {number} [x2] - x2 of
186     the overlay
187     * @param {number} [y2] - y2 of
188     the overlay
189     * @returns {Type.SelectionOverlay}
190     } - This instance
191     * @private
192     */
193     this._setValues = function (x1, y1,
194         x2, y2) {
195         if (x1 !== null) this.x1 = x1;
196         if (y1 !== null) this.y1 = y1;
197         if (x2 !== null) this.x2 = x2;
198         if (y2 !== null) this.y2 = y2;
199         return this;
200     };
201     /**
202     * Sets dimension and position
203     values to th element's style
204     unless they are not different
205     to the current values.
206     */
207     * @param {number} [x1] -
208     Horizontal position of the
209     overlay
210     * @param {number} [y1] - Vertical
211     position of the overlay
212     * @param {number} [x2] - x2 of
213     the overlay
214     * @param {number} [y2] - y2 of
215     the overlay
216     * @returns {Type.SelectionOverlay}
217     } - This instance
218     * @private
219     */
220     this._recalculate = function () {
221         // If x1 or y2 have changed,
222         // recalculate the height
223         if ((y1 !== null && y1 !== this.
224             y1) || (y2 !== null && y2
225             !== this.y2)) {
226             y1 = y1 !== null ? y1 : this.
227                 y1;
228             y2 = y2 !== null ? y2 : this.
229                 y2;
230             this._el.style.height = (y2 -
231                 y1) + 'px';
232         }
233         // If x1 or x2 have changed,
234         // recalculate the width
235         if ((x1 !== null && x1 !== this.
236             x1) || (x2 !== null && x2
237             !== this.x2)) {
238             x1 = x1 !== null ? x1 : this.
239                 x1;
240             x2 = x2 !== null ? x2 : this.
241                 x2;
242             this._el.style.width = (x2 - x1
243                 ) + 'px';
244         }
245         // If y1 has changed, reposition
246         if (y1 !== null && y1 !== this.
247             y1) {
248             this._el.style.top = y1 + '
249                 px';
250         }
251         // If y1 or y2 have changed,
252         // recalculate the height
253         if ((y1 !== null && y1 !== this.
254             y1) || (y2 !== null && y2
255             !== this.y2)) {
256             y1 = y1 !== null ? y1 : this.
257                 y1;
258             y2 = y2 !== null ? y2 : this.
259                 y2;
260             this._el.style.height = (y2 -
261                 y1) + 'px';
262         }
263     };
264 }

```

```

227         return this;
228     };
229     /**
230     * Creates and returns the visible
231     * selection overlay element
232     */
233     * @returns {Element}
234     * @private
235     */
236     this._createElement = function ()
237     {
238         return Type.DomUtilities.
239             addElement('div',
240                 selection');
241     };
242     }.call (Type.SelectionOverlay.
243         prototype);
244     /**
245     * @param {Range} range
246     * @returns {Type.SelectionOverlay}
247     */
248     Type.SelectionOverlay.fromRange =
249         function (range) {
250             var rect = Type.SelectionOverlay.
251                 _getPositionsFromRange(range);
252             return new Type.SelectionOverlay(
253                 rect.left, rect.top, rect.
254                 right, rect.bottom, true,
255                 range.startContainer);
256         };
257     /**
258     *
259     */
260     return this;
261 };
262
263 // @param x
264 // @param y
265 // @returns {Type.SelectionOverlay}
266 // @deprecated
267 //
268 // Type.SelectionOverlay.fromPosition
269 // = function (x, y) {
270 //     var range = document.
271 //         caretRangeFromPoint(x, y);
272 //     return Type.SelectionOverlay.
273 //         fromRange(range);
274 // };
275
276 // Return's the window's horizontal
277 // an vertical scroll positions
278
279 * @todo code duplication to caret.
280 _getScrollPosition
281
282 * @returns {{top: (number), left: (
283     number)}}
284 * @private
285 */
286 Type.SelectionOverlay.
287     _getPositionsFromRange =
288     function (range) {
289         var scroll = Type.SelectionOverlay.
290             _getScrollPosition();
291         var rect = range.getClientRects(
292             [0];
293         if (!rect) {
294             return null;
295         }
296         return {
297             top : rect.top + scroll.top,
298             right : rect.right + scroll.
299                 left,
300             bottom : rect.bottom + scroll.
301                 top,
302             left : rect.left + scroll.left
303         };
304     };
305
306 module.exports = Type.
307     SelectionOverlay;
308
309 Listing 16: selection_overlay.js

```

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6   * @param {Type} type
7   * @constructor
8   */
9  Type.Selection = function (type) {
10     this._init(type);
11 }
12
13 (function () {
14     /**
15      * Resets (removes) the current
16      * selection if there is one,
17      * sets a new anchor at
18      * the given coordinates and sets
19      * up a new selection at the
20      * node and offset found
21      * at the coordinates.
22      * @param {number} x - Absolute
23      * horizontal position on the
24      * document
25      * @param {number} y - Absolute
26      * vertical position on the
27      * document
28      * @returns {Type.Selection} -
29      * This instance
30      */
31     this.beginAt = function (x, y) {
32         this._unselect();
33         this._setAnchor(x, y);
34         return this._startRangeAt(this._
35             anchor.node, this._anchor.
36             offset);
37     };
38
39     /** Will move the end or the start
40      * of the selection to the node
41      * and offset found at
42
43     33  * the given coordinates. Whether
44     34  * the start or the end will be
45     35  * moved depends on
46     36  * whether the coordinates are on
47     37  * top / left of this selection
48     38  * 's anchor or below /
49     39  * right of it.
50     40  * @param {number} x - Absolute
51     41  * horizontal position on the
52     42  * document
53     43  * @param {number} y - Absolute
54     44  * vertical position on the
55     45  * document
56     46  * @returns {Type.Selection} -
57     47  * This instance
58
59     48  this._moveTo = function (x, y) {
60     49  var range = Type.Range.fromPoint
61     50  (x, y);
62     51  this._addElement(range);
63     52  if (x < this._anchor.x || y <
64     53  this._anchor.y) {
65     54  this._moveStartTo(range);
66     55  endContainer, range;
67     56  endOffset);
68     57  } else {
69     58  this._moveEndTo(range);
70     59  endContainer, range;
71     60  endOffset);
72     61  }
73     62  return this;
74 }
75
76 /** Returns the contents of the
77  * selection or null if there
78  * is no selection
79  * @returns {DocumentFragment}
80  */
81 this._getContent = function () {
82     return this._range ? this._range
83     .cloneContents() : null;
84
85     65  * @param x
86     66  * @param y
87     67  * @returns {*}
88     68  */
89     this.selectWordAt = function (x, y)
90     {
91         var charAtStart, charAtEnd,
92         whitespace = new RegExp('\\s')
93         ,
94         endLength = this._range.
95         endContainer.nodeValue.
96         length,
97         startOffset = this._range.
98         startOffset,
99         endOffset = this._range.
100        endOffset,
101        startFound = false,
102        endFound = false;
103
104        this._beginAt(x, y);
105
106        do {
107            charAtStart = this._range.
108            startContainer.nodeValue.
109            charAt(this._range.
110            startOffset - 1);
111
112            if (startOffset > 1 && !

```

```

116         whitespace.test(
117             charAtStart)) {
118             if (startOffset > 1) {
119                 startOffset -= 1;
120                 this._range.setStart(this._
121                     _range.startContainer
122                     , startOffset);
123             }
124             } else {
125                 startFound = true;
126             }
127         } while (!startFound);
128     do {
129         charAtEnd = this._range.
130             endContainer.nodeValue.
131             charAt(this._range.
132                 endOffset);
133     } if (endOffset < endLength && !
134         whitespace.test(charAtEnd
135             )) {
136         if (endOffset < endLength) {
137             endOffset += 1;
138             this._range.setEnd(this._
139                 _range.endContainer,
140                 endOffset);
141         } else {
142             endFound = true;
143         }
144     } while (!endFound);
145     this._imitateRangeAppending();
146     return this;
147 };
148 /**
149  * Removes all selection overlays
150  * and resets internal
151  * variables.
152
153     over the currently selected
154     text.
155     * @returns {Type.Range}
156     */
157     this._range = function () {
158         return Type.Range.fromRange(this
159             ._range);
160     };
161 /**
162  * Returns the {Range} this
163  * selection spans over or null
164  * if nothing has been
165  * selected yet.
166  * @returns {Range|null}
167     */
168     this.getNativeRange = function ()
169     {
170         return this._range;
171     };
172 /**
173  * Returns the start node and
174  * offset of this selection.
175  * @returns {{node: Node, offset:
176     number}|null}
177     */
178     this.getStart = function () {
179         if (this._range) {
180             return {node: this._range.
181                 startContainer, offset:
182                 this._range.startOffset};
183         }
184         return null;
185     };
186 /**
187  * Returns the end node and offset
188  * of this selection.
189  * @returns {{node: Node, offset:
190     number}|null}
191     */
192     this.getEnd = function () {
193         if (this._range) {
194

```

```

182         return {node: this._range,
183               endContainer, offset :
184               this._range.endOffset};
185     }
186     return null;
187 }
188 /**
189  * Returns whether or not this
190  * selection is visible. By
191  * checking if there currently
192  * are any overlays and if the
193  * first overlay is actually
194  * visible. There should be
195  * no case where there are visible
196  * overlays but the first
197  * overlay wouldn't be visible,
198  * so this is a quick and
199  * performant way to check for
200  * the selection's visibility.
201  *
202  * @returns {boolean} - True if
203  * selection is hidden, false
204  * if there is a selection
205  */
206 this.collapsed = function () {
207     return !this._overlays.length ||
208         !this._overlays[0].visible
209         ();
210 };
211 /**
212  * Alias method for select() for
213  * better code readability. For
214  * initialization
215  * all variables should be set to
216  * their default values. This
217  * is what select
218  * does for us.
219  *
220  * @param {Type} type
221  * @returns {Type.Selection} -
222  * This instance
223  * @private
224  */
225 this._init = function (type) {
226     return {node: type.getRoot();
227           return this.unselect();
228         };
229     /**
230      * Creates a new {Range}, which
231      * will be the basis for
232      * drawing and this selection.
233      * todo Use {Type.Range}? Should
234      * be cool if we don't use
235      * getRects or we make Type.
236      * Range more performant
237      *
238      * @param {Node} node - The text
239      * node that the selection
240      * should start in
241      * @param {number} offset - The
242      * offset in the text node that
243      * the selection should start
244      * in
245      * @returns {Type.Selection} -
246      * This instance
247      */
248     this._startRangeAt = function (
249         node, offset) {
250         this._range = window.document.
251             createRange();
252         this._range.setStart(node,
253             offset);
254         this._range.setEnd(node, offset)
255         ;
256         return this;
257     };
258     /**
259      * Sets the anchor node, offset
260      * and position in this screen
261      * for this selection.
262      * When a user draws a selection,
263      * what is being selected
264      * depends on whether he /
265      * she moves his / her mouse
266      * before or behind the point
267      * he / she started to draw
268      * the selection. The information
269      * in the anchor needs to be
270      * saved to implement
271      * this behaviour.
272     */
273     this._moveEndTo = function (node,
274         offset) {
275         this._range.setStart(this.
276             _anchor.node, this._anchor.
277             _offset);
278         this._range.setEnd(node, offset)
279         ;
280         this._imitateRangeAppending();
281         return this;
282     };
283     /**
284      * Sets the anchor node, offset
285      * and position in this screen
286      * for this selection.
287      * When a user draws a selection,
288      * what is being selected
289      * depends on whether he /
290      * she moves his / her mouse
291      * before or behind the point
292      * he / she started to draw
293      * the selection. The information
294      * in the anchor needs to be
295      * saved to implement
296      * this behaviour.
297     */
298     this._moveStartTo = function (node

```



```

260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286

287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312

313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339

    * @param {number} x - Absolute
    horizontal position on the
    document
    * @param {number} y - Absolute
    vertical position on the
    document
    * @returns {Type.Selection} -
    This instance
    * @private
    this._setAnchor = function (x, y)
    {
        var range = Type.Range.fromPoint
        (x, y);
        this._anchor = {x: x, y: y, node
        : range.startContainer,
        offset: range.startOffset};
        this._addElement(this._anchor.
        node);
        return this;
    };
    /**
    * Creates {Type.SelectionOverlay}
    s that mimic the appearance
    of
    * the selection as drawn by {this
    ._range}
    * @returns {Type.Selection} -
    This instance
    * @private
    this._imitateRangePrepending =
    function () {
        // Required variables
        var rects = Type.Range.
        getClientRects(this._range)
        , // this._range.
        getClientRects(),
        draw,
        overlay,
        i;

        * @returns {Type.Selection} -
        This instance
        * @private
        this._imitateRangeAppending =
        function () {
            // Required variables
            var rects = this._range.
            getClientRects(),
            draw,
            overlay,
            i;

            // Resize and add overlays to
            match the range's rects
            for (i = 0; i < rects.length; i
            += 1) {
                if (this._overlays[i]) {
                    this._overlays[i].set(rects[i]
                    .left, rects[i].top,
                    rects[i].right, rects[i]
                    .bottom);
                } else {
                    draw = !this.
                    matchesElementDimensions
                    (rects[i]);
                    overlay = new Type.
                    SelectionOverlay(rects[i]
                    .left, rects[i].top,
                    rects[i].right, rects[i]
                    .bottom, draw);
                    this._overlays.unshift(
                    overlay);
                }
            }
            // Remove overlays prepending
            the current range's rects
            while (this._overlays.length >
            rects.length) {
                this._overlays.shift().remove
                ();
            }
            // Chaining
            return this;
        };
        /**
        * Creates {Type.SelectionOverlay}
        s that mimic the appearance
        of
        * the selection as drawn by {this
        ._range}
        *
        */
    }

```

```

340 // Chaining
341 return this;
342 };
343 /**
344 * Todo scrolling
345 *
346 * @param {Node|Element} el - An
347 *   element or a text node
348 * @returns {Type.Selection} -
349 *   This instance
350 * @private
351 */
352 this._addElement = function (el) {
353   var rect, key;
354   el = el.nodeType === 3 ? el.
355     parentNode : el;
356   rect = el.getBoundingClientRect
357     ();
358   key = this._stringifyRect(rect);
359   this._elements[key] = rect;
360   return this;
361 };
362 /**
363 *
364 * @param {ClientRect} rect
365 * @private
366 */
367 this._matchesElementDimensions =
368   function (rect) {
369     var key = this._stringifyRect(
370       rect);
371     return this._elements.
372       hasOwnProperty(key);
373   };
374 /**
375 * Removes all selection overlays
376 *
377 * @returns {Type.Selection} -
378 *   This instance
379 * @private
380 */
381 this._removeOverlays = function ()
382 {
383   var i;
384   this._overlays = this._overlays
385     || [];
386   for (i = 0; i < this._overlays.
387     length; i += 1) {
388     this._overlays[i].remove();
389   }
390 }
391
392 //this._content = type.getContent
393 ();
394 this._content = new Type.Content(
395   type);
396 this._writer = type.getWriter();
397 this._caret = type.getCaret();
398 this._selection = this._type.
399   getSelection();
400 this._el = this._createElement();
401 this._elStyle = this._el.style;
402 this._caretStyle = this._caret.
403   caretEl.style;
404 this._loadFilters();
405
406 //this._bindEvents();
407 };
408 (function () {
409   /**
410    * Adds a filter to the input
411    * pipeline
412    *
413    * @param {String} name - An
414    *   identifier for the filter
415    * @param {Object} filter - A
416    *   filter
417
418   this._overlays = [];
419   return this;
420 }
421 /**
422 * @param {ClientRect} rect
423 * @returns {string}
424 * @private
425 */
426 this._stringifyRect = function (
427   rect) {
428   var top = rect.top.toString(),
429       left = rect.left.toString(),
430       bottom = rect.bottom.toString()
431       (),
432       right = rect.right.toString()
433       ();
434   return top + left + bottom +
435     right;
436 }
437 ).call(Type.Selection.prototype);
438 module.exports = Type.Selection;
439
440 Listing 17: selection.js
441
442 this._bindEvents();
443 };
444 (function () {
445   /**
446    * Adds a filter to the input
447    * pipeline
448    *
449    * @param {String} name - An
450    *   identifier for the filter
451    * @param {Object} filter - A
452    *   filter
453
454   this._content = type.getContent
455     ();
456   this._content = new Type.Content(
457     type);
458   this._writer = type.getWriter();
459   this._caret = type.getCaret();
460   this._selection = this._type.
461     getSelection();
462   this._el = this._createElement();
463   this._elStyle = this._el.style;
464   this._caretStyle = this._caret.
465     caretEl.style;
466   this._loadFilters();
467
468   //this._content = type.getContent
469   ();
470   this._content = new Type.Content(
471     type);
472   this._writer = type.getWriter();
473   this._caret = type.getCaret();
474   this._selection = this._type.
475     getSelection();
476   this._el = this._createElement();
477   this._elStyle = this._el.style;
478   this._caretStyle = this._caret.
479     caretEl.style;
480   this._loadFilters();
481
482   //this._content = type.getContent
483   ();
484   this._content = new Type.Content(
485     type);
486   this._writer = type.getWriter();
487   this._caret = type.getCaret();
488   this._selection = this._type.
489     getSelection();
490   this._el = this._createElement();
491   this._elStyle = this._el.style;
492   this._caretStyle = this._caret.
493     caretEl.style;
494   this._loadFilters();
495
496   //this._content = type.getContent
497   ();
498   this._content = new Type.Content(
499     type);
500   this._writer = type.getWriter();
501   this._caret = type.getCaret();
502   this._selection = this._type.
503     getSelection();
504   this._el = this._createElement();
505   this._elStyle = this._el.style;
506   this._caretStyle = this._caret.
507     caretEl.style;
508   this._loadFilters();
509
510   //this._content = type.getContent
511   ();
512   this._content = new Type.Content(
513     type);
514   this._writer = type.getWriter();
515   this._caret = type.getCaret();
516   this._selection = this._type.
517     getSelection();
518   this._el = this._createElement();
519   this._elStyle = this._el.style;
520   this._caretStyle = this._caret.
521     caretEl.style;
522   this._loadFilters();
523
524   //this._content = type.getContent
525   ();
526   this._content = new Type.Content(
527     type);
528   this._writer = type.getWriter();
529   this._caret = type.getCaret();
530   this._selection = this._type.
531     getSelection();
532   this._el = this._createElement();
533   this._elStyle = this._el.style;
534   this._caretStyle = this._caret.
535     caretEl.style;
536   this._loadFilters();
537
538   //this._content = type.getContent
539   ();
540   this._content = new Type.Content(
541     type);
542   this._writer = type.getWriter();
543   this._caret = type.getCaret();
544   this._selection = this._type.
545     getSelection();
546   this._el = this._createElement();
547   this._elStyle = this._el.style;
548   this._caretStyle = this._caret.
549     caretEl.style;
550   this._loadFilters();
551
552   //this._content = type.getContent
553   ();
554   this._content = new Type.Content(
555     type);
556   this._writer = type.getWriter();
557   this._caret = type.getCaret();
558   this._selection = this._type.
559     getSelection();
560   this._el = this._createElement();
561   this._elStyle = this._el.style;
562   this._caretStyle = this._caret.
563     caretEl.style;
564   this._loadFilters();
565
566   //this._content = type.getContent
567   ();
568   this._content = new Type.Content(
569     type);
570   this._writer = type.getWriter();
571   this._caret = type.getCaret();
572   this._selection = this._type.
573     getSelection();
574   this._el = this._createElement();
575   this._elStyle = this._el.style;
576   this._caretStyle = this._caret.
577     caretEl.style;
578   this._loadFilters();
579
580   //this._content = type.getContent
581   ();
582   this._content = new Type.Content(
583     type);
584   this._writer = type.getWriter();
585   this._caret = type.getCaret();
586   this._selection = this._type.
587     getSelection();
588   this._el = this._createElement();
589   this._elStyle = this._el.style;
590   this._caretStyle = this._caret.
591     caretEl.style;
592   this._loadFilters();
593
594   //this._content = type.getContent
595   ();
596   this._content = new Type.Content(
597     type);
598   this._writer = type.getWriter();
599   this._caret = type.getCaret();
600   this._selection = this._type.
601     getSelection();
602   this._el = this._createElement();
603   this._elStyle = this._el.style;
604   this._caretStyle = this._caret.
605     caretEl.style;
606   this._loadFilters();
607
608   //this._content = type.getContent
609   ();
610   this._content = new Type.Content(
611     type);
612   this._writer = type.getWriter();
613   this._caret = type.getCaret();
614   this._selection = this._type.
615     getSelection();
616   this._el = this._createElement();
617   this._elStyle = this._el.style;
618   this._caretStyle = this._caret.
619     caretEl.style;
620   this._loadFilters();
621
622   //this._content = type.getContent
623   ();
624   this._content = new Type.Content(
625     type);
626   this._writer = type.getWriter();
627   this._caret = type.getCaret();
628   this._selection = this._type.
629     getSelection();
630   this._el = this._createElement();
631   this._elStyle = this._el.style;
632   this._caretStyle = this._caret.
633     caretEl.style;
634   this._loadFilters();
635
636   //this._content = type.getContent
637   ();
638   this._content = new Type.Content(
639     type);
640   this._writer = type.getWriter();
641   this._caret = type.getCaret();
642   this._selection = this._type.
643     getSelection();
644   this._el = this._createElement();
645   this._elStyle = this._el.style;
646   this._caretStyle = this._caret.
647     caretEl.style;
648   this._loadFilters();
649
650   //this._content = type.getContent
651   ();
652   this._content = new Type.Content(
653     type);
654   this._writer = type.getWriter();
655   this._caret = type.getCaret();
656   this._selection = this._type.
657     getSelection();
658   this._el = this._createElement();
659   this._elStyle = this._el.style;
660   this._caretStyle = this._caret.
661     caretEl.style;
662   this._loadFilters();
663
664   //this._content = type.getContent
665   ();
666   this._content = new Type.Content(
667     type);
668   this._writer = type.getWriter();
669   this._caret = type.getCaret();
670   this._selection = this._type.
671     getSelection();
672   this._el = this._createElement();
673   this._elStyle = this._el.style;
674   this._caretStyle = this._caret.
675     caretEl.style;
676   this._loadFilters();
677
678   //this._content = type.getContent
679   ();
680   this._content = new Type.Content(
681     type);
682   this._writer = type.getWriter();
683   this._caret = type.getCaret();
684   this._selection = this._type.
685     getSelection();
686   this._el = this._createElement();
687   this._elStyle = this._el.style;
688   this._caretStyle = this._caret.
689     caretEl.style;
690   this._loadFilters();
691
692   //this._content = type.getContent
693   ();
694   this._content = new Type.Content(
695     type);
696   this._writer = type.getWriter();
697   this._caret = type.getCaret();
698   this._selection = this._type.
699     getSelection();
700   this._el = this._createElement();
701   this._elStyle = this._el.style;
702   this._caretStyle = this._caret.
703     caretEl.style;
704   this._loadFilters();
705
706   //this._content = type.getContent
707   ();
708   this._content = new Type.Content(
709     type);
710   this._writer = type.getWriter();
711   this._caret = type.getCaret();
712   this._selection = this._type.
713     getSelection();
714   this._el = this._createElement();
715   this._elStyle = this._el.style;
716   this._caretStyle = this._caret.
717     caretEl.style;
718   this._loadFilters();
719
720   //this._content = type.getContent
721   ();
722   this._content = new Type.Content(
723     type);
724   this._writer = type.getWriter();
725   this._caret = type.getCaret();
726   this._selection = this._type.
727     getSelection();
728   this._el = this._createElement();
729   this._elStyle = this._el.style;
730   this._caretStyle = this._caret.
731     caretEl.style;
732   this._loadFilters();
733
734   //this._content = type.getContent
735   ();
736   this._content = new Type.Content(
737     type);
738   this._writer = type.getWriter();
739   this._caret = type.getCaret();
740   this._selection = this._type.
741     getSelection();
742   this._el = this._createElement();
743   this._elStyle = this._el.style;
744   this._caretStyle = this._caret.
745     caretEl.style;
746   this._loadFilters();
747
748   //this._content = type.getContent
749   ();
750   this._content = new Type.Content(
751     type);
752   this._writer = type.getWriter();
753   this._caret = type.getCaret();
754   this._selection = this._type.
755     getSelection();
756   this._el = this._createElement();
757   this._elStyle = this._el.style;
758   this._caretStyle = this._caret.
759     caretEl.style;
760   this._loadFilters();
761
762   //this._content = type.getContent
763   ();
764   this._content = new Type.Content(
765     type);
766   this._writer = type.getWriter();
767   this._caret = type.getCaret();
768   this._selection = this._type.
769     getSelection();
770   this._el = this._createElement();
771   this._elStyle = this._el.style;
772   this._caretStyle = this._caret.
773     caretEl.style;
774   this._loadFilters();
775
776   //this._content = type.getContent
777   ();
778   this._content = new Type.Content(
779     type);
780   this._writer = type.getWriter();
781   this._caret = type.getCaret();
782   this._selection = this._type.
783     getSelection();
784   this._el = this._createElement();
785   this._elStyle = this._el.style;
786   this._caretStyle = this._caret.
787     caretEl.style;
788   this._loadFilters();
789
790   //this._content = type.getContent
791   ();
792   this._content = new Type.Content(
793     type);
794   this._writer = type.getWriter();
795   this._caret = type.getCaret();
796   this._selection = this._type.
797     getSelection();
798   this._el = this._createElement();
799   this._elStyle = this._el.style;
800   this._caretStyle = this._caret.
801     caretEl.style;
802   this._loadFilters();
803
804   //this._content = type.getContent
805   ();
806   this._content = new Type.Content(
807     type);
808   this._writer = type.getWriter();
809   this._caret = type.getCaret();
810   this._selection = this._type.
811     getSelection();
812   this._el = this._createElement();
813   this._elStyle = this._el.style;
814   this._caretStyle = this._caret.
815     caretEl.style;
816   this._loadFilters();
817
818   //this._content = type.getContent
819   ();
820   this._content = new Type.Content(
821     type);
822   this._writer = type.getWriter();
823   this._caret = type.getCaret();
824   this._selection = this._type.
825     getSelection();
826   this._el = this._createElement();
827   this._elStyle = this._el.style;
828   this._caretStyle = this._caret.
829     caretEl.style;
830   this._loadFilters();
831
832   //this._content = type.getContent
833   ();
834   this._content = new Type.Content(
835     type);
836   this._writer = type.getWriter();
837   this._caret = type.getCaret();
838   this._selection = this._type.
839     getSelection();
840   this._el = this._createElement();
841   this._elStyle = this._el.style;
842   this._caretStyle = this._caret.
843     caretEl.style;
844   this._loadFilters();
845
846   //this._content = type.getContent
847   ();
848   this._content = new Type.Content(
849     type);
850   this._writer = type.getWriter();
851   this._caret = type.getCaret();
852   this._selection = this._type.
853     getSelection();
854   this._el = this._createElement();
855   this._elStyle = this._el.style;
856   this._caretStyle = this._caret.
857     caretEl.style;
858   this._loadFilters();
859
860   //this._content = type.getContent
861   ();
862   this._content = new Type.Content(
863     type);
864   this._writer = type.getWriter();
865   this._caret = type.getCaret();
866   this._selection = this._type.
867     getSelection();
868   this._el = this._createElement();
869   this._elStyle = this._el.style;
870   this._caretStyle = this._caret.
871     caretEl.style;
872   this._loadFilters();
873
874   //this._content = type.getContent
875   ();
876   this._content = new Type.Content(
877     type);
878   this._writer = type.getWriter();
879   this._caret = type.getCaret();
880   this._selection = this._type.
881     getSelection();
882   this._el = this._createElement();
883   this._elStyle = this._el.style;
884   this._caretStyle = this._caret.
885     caretEl.style;
886   this._loadFilters();
887
888   //this._content = type.getContent
889   ();
890   this._content = new Type.Content(
891     type);
892   this._writer = type.getWriter();
893   this._caret = type.getCaret();
894   this._selection = this._type.
895     getSelection();
896   this._el = this._createElement();
897   this._elStyle = this._el.style;
898   this._caretStyle = this._caret.
899     caretEl.style;
900   this._loadFilters();
901
902   //this._content = type.getContent
903   ();
904   this._content = new Type.Content(
905     type);
906   this._writer = type.getWriter();
907   this._caret = type.getCaret();
908   this._selection = this._type.
909     getSelection();
910   this._el = this._createElement();
911   this._elStyle = this._el.style;
912   this._caretStyle = this._caret.
913     caretEl.style;
914   this._loadFilters();
915
916   //this._content = type.getContent
917   ();
918   this._content = new Type.Content(
919     type);
920   this._writer = type.getWriter();
921   this._caret = type.getCaret();
922   this._selection = this._type.
923     getSelection();
924   this._el = this._createElement();
925   this._elStyle = this._el.style;
926   this._caretStyle = this._caret.
927     caretEl.style;
928   this._loadFilters();
929
930   //this._content = type.getContent
931   ();
932   this._content = new Type.Content(
933     type);
934   this._writer = type.getWriter();
935   this._caret = type.getCaret();
936   this._selection = this._type.
937     getSelection();
938   this._el = this._createElement();
939   this._elStyle = this._el.style;
940   this._caretStyle = this._caret.
941     caretEl.style;
942   this._loadFilters();
943
944   //this._content = type.getContent
945   ();
946   this._content = new Type.Content(
947     type);
948   this._writer = type.getWriter();
949   this._caret = type.getCaret();
950   this._selection = this._type.
951     getSelection();
952   this._el = this._createElement();
953   this._elStyle = this._el.style;
954   this._caretStyle = this._caret.
955     caretEl.style;
956   this._loadFilters();
957
958   //this._content = type.getContent
959   ();
960   this._content = new Type.Content(
961     type);
962   this._writer = type.getWriter();
963   this._caret = type.getCaret();
964   this._selection = this._type.
965     getSelection();
966   this._el = this._createElement();
967   this._elStyle = this._el.style;
968   this._caretStyle = this._caret.
969     caretEl.style;
970   this._loadFilters();
971
972   //this._content = type.getContent
973   ();
974   this._content = new Type.Content(
975     type);
976   this._writer = type.getWriter();
977   this._caret = type.getCaret();
978   this._selection = this._type.
979     getSelection();
980   this._el = this._createElement();
981   this._elStyle = this._el.style;
982   this._caretStyle = this._caret.
983     caretEl.style;
984   this._loadFilters();
985
986   //this._content = type.getContent
987   ();
988   this._content = new Type.Content(
989     type);
989

```

```

36 * @returns {Type.Input}
37 */
38 this.addFilter = function (name,
39 filter) {
40     this._filters = this._filters ||
41     {};
42     this._filters[name] = filter;
43     return this;
44 };
45 /**
46 * Removes a filter from the input
47 pipeline
48 *
49 * @param {String} name - An
50 identifier for the filter
51 * @returns {Type.Input}
52 */
53 this.removeFilter = function (name)
54 {
55     this._filters = this._filters ||
56     {};
57     if (this._filters.name) {
58         delete this._filters.name;
59         return this;
60     }
61     /**
62 * Getter for this instance's
63 content.
64 * @returns {Type.Content}
65 */
66 this.getContent = function () {
67     return this._content;
68 };
69 /**
70 *
71 * @returns {Type.Input}
72 * @private
73 */
74 this._loadFilters = function () {
75     this._filters = this._filters ||
76     {};
77     this._filters.undo = new Type.
78     Input.Filter.Undo(this.
79     _type, this);
80     this._filters.cmd = new Type.
81     Input.Filter.Command(this.
82     _type, this);
83     this._filters.caret = new Type.
84     Input.Filter.Caret(this.
85     _type, this);
86     this._filters.remove = new Type.
87     Input.Filter.Remove(this.
88     _type, this);
89     this._filters.lineBreaks = new
90     Type.Input.Filter.
91     LineBreaks(this._type, this
92     );
93     return this;
94 };
95 /**
96 * Binds events on type's root
97 element to catch keyboard
98 * and mouse input.
99 *
100 * @returns {Type.Input}
101 * @private
102 */
103 this._bindEvents = function () {
104     this._bindKeyDownEvents();
105     this._bindInputEvents = function
106     (
107         , function (e) {
108             this._onInput(e);
109             this._bind(this, false);
110             return this;
111         }
112     );
113     /**
114 * Todo Legacy Internet Explorer
115 and attachEvent https://
116 developer.mozilla.org/en-US/
117 docs/Web/API/EventTarget/
118 addEventListener
119 */
120 * @returns {Type.Input}
121 * @private
122 */
123 this._bindMouseEvents = function
124 (
125     , function (e) {
126         developer.mozilla.org/en-US/
127 docs/Web/API/EventTarget/
128 addEventListener
129 }
130 );
131

```

```

132     () {
133     var self = this;
134     function dragSelection(e) {
135     self._selection.moveTo(e,
136     clientX, e.clientY);
137     }
138     function stopDraggingSelection ()
139     {
140     document.removeEventListener('
141     mousemove', dragSelection
142     , false);
143     document.removeEventListener('
144     mouseup',
145     stopDraggingSelection ,
146     false);
147     self._el.innerHTML = '';
148     self._el.appendChild(self,
149     _selection.getContent());
150     document.execCommand('
151     selectAll', false, null);
152     }
153     function startDraggingSelection (
154     e) {
155     if (e.which === 1) {
156     e.preventDefault();
157     self._caret._hide();
158     self._selection.beginAt(e,
159     clientX, e.clientY);
160     document.addEventListener('
161     mousemove',
162     dragSelection, false);
163     document.addEventListener('
164     mouseup',
165     stopDraggingSelection ,
166     false);
167     }
168     }
169     function stopDraggingSelection (e) {
170     document.removeEventListener('
171     mousemove', dragSelection
172     , false);
173     document.removeEventListener('
174     mouseup',
175     stopDraggingSelection ,
176     false);
177     self._el.innerHTML = '';
178     self._el.appendChild(self,
179     _selection.getContent());
180     document.execCommand('
181     selectAll', false, null);
182     }
183     }
184     }
185     }
186     }
187     }
188     }
189     }
190     }
191     }
192     }
193     }
194     }
195     }
196     }
197     }
198     }
199     }
200     }
201     }
202     }
203     }
204     }
205     }
206     }
207     }
208     }
209     }
210     }
211     }
212     }
213     }
214     }
215     }
216     }
217     }
218     }
219     }
220     }
221     }
222     }
223     }
224     }
225     }
226     }
227     }
228     }
229     }
230     }
231     }
232     }
233     }
234     }
235     }
236     }
237     }
238     }
239     }
240     }
241     }
242     }
243     }
244     }
245     }
246     }
247     }
248     }
249     }
250     }
251     }
252     }
253     }
254     }
255     }
256     }
257     }
258     }
259     }
260     }
261     }
262     }
263     }
264     }
265     }
266     }
267     }
268     }
269     }
270     }
271     }
272     }
273     }
274     }
275     }
276     }
277     }
278     }
279     }
280     }
281     }
282     }
283     }
284     }
285     }
286     }
287     }
288     }
289     }
290     }
291     }
292     }
293     }
294     }
295     }
296     }
297     }
298     }
299     }
300     }
301     }
302     }
303     }
304     }
305     }
306     }
307     }
308     }
309     }
310     }
311     }
312     }
313     }
314     }
315     }
316     }
317     }
318     }
319     }
320     }
321     }
322     }
323     }
324     }
325     }
326     }
327     }
328     }
329     }
330     }
331     }
332     }
333     }
334     }
335     }
336     }
337     }
338     }
339     }
340     }
341     }
342     }
343     }
344     }
345     }
346     }
347     }
348     }
349     }
350     }
351     }
352     }
353     }
354     }
355     }
356     }
357     }
358     }
359     }
360     }
361     }
362     }
363     }
364     }
365     }
366     }
367     }
368     }
369     }
370     }
371     }
372     }
373     }
374     }
375     }
376     }
377     }
378     }
379     }
380     }
381     }
382     }
383     }
384     }
385     }
386     }
387     }
388     }
389     }
390     }
391     }
392     }
393     }
394     }
395     }
396     }
397     }
398     }
399     }
400     }
401     }
402     }
403     }
404     }
405     }
406     }
407     }
408     }
409     }
410     }
411     }
412     }
413     }
414     }
415     }
416     }
417     }
418     }
419     }
420     }
421     }
422     }
423     }
424     }
425     }
426     }
427     }
428     }
429     }
430     }
431     }
432     }
433     }
434     }
435     }
436     }
437     }
438     }
439     }
440     }
441     }
442     }
443     }
444     }
445     }
446     }
447     }
448     }
449     }
450     }
451     }
452     }
453     }
454     }
455     }
456     }
457     }
458     }
459     }
460     }
461     }
462     }
463     }
464     }
465     }
466     }
467     }
468     }
469     }
470     }
471     }
472     }
473     }
474     }
475     }
476     }
477     }
478     }
479     }
480     }
481     }
482     }
483     }
484     }
485     }
486     }
487     }
488     }
489     }
490     }
491     }
492     }
493     }
494     }
495     }
496     }
497     }
498     }
499     }
500     }
501     }
502     }
503     }
504     }
505     }
506     }
507     }
508     }
509     }
510     }
511     }
512     }
513     }
514     }
515     }
516     }
517     }
518     }
519     }
520     }
521     }
522     }
523     }
524     }
525     }
526     }
527     }
528     }
529     }
530     }
531     }
532     }
533     }
534     }
535     }
536     }
537     }
538     }
539     }
540     }
541     }
542     }
543     }
544     }
545     }
546     }
547     }
548     }
549     }
550     }
551     }
552     }
553     }
554     }
555     }
556     }
557     }
558     }
559     }
560     }
561     }
562     }
563     }
564     }
565     }
566     }
567     }
568     }
569     }
570     }
571     }
572     }
573     }
574     }
575     }
576     }
577     }
578     }
579     }
580     }
581     }
582     }
583     }
584     }
585     }
586     }
587     }
588     }
589     }
590     }
591     }
592     }
593     }
594     }
595     }
596     }
597     }
598     }
599     }
600     }
601     }
602     }
603     }
604     }
605     }
606     }
607     }
608     }
609     }
610     }
611     }
612     }
613     }
614     }
615     }
616     }
617     }
618     }
619     }
620     }
621     }
622     }
623     }
624     }
625     }
626     }
627     }
628     }
629     }
630     }
631     }
632     }
633     }
634     }
635     }
636     }
637     }
638     }
639     }
640     }
641     }
642     }
643     }
644     }
645     }
646     }
647     }
648     }
649     }
650     }
651     }
652     }
653     }
654     }
655     }
656     }
657     }
658     }
659     }
660     }
661     }
662     }
663     }
664     }
665     }
666     }
667     }
668     }
669     }
670     }
671     }
672     }
673     }
674     }
675     }
676     }
677     }
678     }
679     }
680     }
681     }
682     }
683     }
684     }
685     }
686     }
687     }
688     }
689     }
690     }
691     }
692     }
693     }
694     }
695     }
696     }
697     }
698     }
699     }
700     }
701     }
702     }
703     }
704     }
705     }
706     }
707     }
708     }
709     }
710     }
711     }
712     }
713     }
714     }
715     }
716     }
717     }
718     }
719     }
720     }
721     }
722     }
723     }
724     }
725     }
726     }
727     }
728     }
729     }
730     }
731     }
732     }
733     }
734     }
735     }
736     }
737     }
738     }
739     }
740     }
741     }
742     }
743     }
744     }
745     }
746     }
747     }
748     }
749     }
750     }
751     }
752     }
753     }
754     }
755     }
756     }
757     }
758     }
759     }
760     }
761     }
762     }
763     }
764     }
765     }
766     }
767     }
768     }
769     }
770     }
771     }
772     }
773     }
774     }
775     }
776     }
777     }
778     }
779     }
780     }
781     }
782     }
783     }
784     }
785     }
786     }
787     }
788     }
789     }
790     }
791     }
792     }
793     }
794     }
795     }
796     }
797     }
798     }
799     }
800     }
801     }
802     }
803     }
804     }
805     }
806     }
807     }
808     }
809     }
810     }
811     }
812     }
813     }
814     }
815     }
816     }
817     }
818     }
819     }
820     }
821     }
822     }
823     }
824     }
825     }
826     }
827     }
828     }
829     }
830     }
831     }
832     }
833     }
834     }
835     }
836     }
837     }
838     }
839     }
840     }
841     }
842     }
843     }
844     }
845     }
846     }
847     }
848     }
849     }
850     }
851     }
852     }
853     }
854     }
855     }
856     }
857     }
858     }
859     }
860     }
861     }
862     }
863     }
864     }
865     }
866     }
867     }
868     }
869     }
870     }
871     }
872     }
873     }
874     }
875     }
876     }
877     }
878     }
879     }
880     }
881     }
882     }
883     }
884     }
885     }
886     }
887     }
888     }
889     }
890     }
891     }
892     }
893     }
894     }
895     }
896     }
897     }
898     }
899     }
900     }
901     }
902     }
903     }
904     }
905     }
906     }
907     }
908     }
909     }
910     }
911     }
912     }
913     }
914     }
915     }
916     }
917     }
918     }
919     }
920     }
921     }
922     }
923     }
924     }
925     }
926     }
927     }
928     }
929     }
930     }
931     }
932     }
933     }
934     }
935     }
936     }
937     }
938     }
939     }
940     }
941     }
942     }
943     }
944     }
945     }
946     }
947     }
948     }
949     }
950     }
951     }
952     }
953     }
954     }
955     }
956     }
957     }
958     }
959     }
960     }
961     }
962     }
963     }
964     }
965     }
966     }
967     }
968     }
969     }
970     }
971     }
972     }
973     }
974     }
975     }
976     }
977     }
978     }
979     }
980     }
981     }
982     }
983     }
984     }
985     }
986     }
987     }
988     }
989     }
990     }
991     }
992     }
993     }
994     }
995     }
996     }
997     }
998     }
999     }
1000    }

```

```

213         length > 2) {
214             this._type.trigger('paste',
215                 [inputEvent]);
216         } else {
217             this._type.trigger('input',
218                 [inputEvent]);
219         }
220     }
221     return inputEvent;
222 };
223 /**
224  * @param filter
225  * @param {Type.Events.Input} e
226  * @returns {Type.Events.Input}
227  * @private
228 */
229 this._processFilter = function (
230     filter, e) {
231     var func = filter.keys[e.key];
232     if (func) {
233         filter[func](e);
234     }
235     if (!e.canceled && filter.keys.
236         all) {
237         filter[filter.keys.all](e)
238     }
239     return e;
240 };
241 /**
242  * @param {InputEvent} e
243  * @returns {Type.Input}
244  * @private
245 */
246 this._onInput = function (e) {
247     this._content.insert(this._caret.
248         .textNode, this._caret.
249         .offset, this._el.
250         .textContent);
251     this._caret._setOffset(this.
252         _caret.offset + this._el.
253         .textContent.length);
254     // todo better api
255     this._el.innerHTML = '';
256     return this;
257 };
258 /**
259  * @param x
260  * @param y
261  * @returns {*}
262  * @private
263 */
264 this._moveCaretToMousePosition =
265     function (x, y) {
266         var range = Type.Range.fromPoint
267             (x, y);
268         if (range.startContainer.
269             nodeType === 3) {
270             this._caret.moveTo(range.
271                 startContainer, range.
272                 startOffset);
273             this._caret._blink();
274         }
275         return this;
276     };
277 /**
278  * @param x
279  * @param y
280  * @returns {*}
281  * @private
282 */
283 this._moveElToPosition = function (
284     x, y) {
285     this._el.style.left = x + 'px';
286     this._el.style.top = y + 'px';
287     return this;
288 };
289 /**
290  * @param {Type.Events.Input} e
291  * @returns {Type.Events.Input}
292  * @private
293 */
294 this._focusInput = function (sync)
295     {
296         if (sync) {
297             this._el.focus();
298         } else {
299             window.setTimeout(function () {
300                 this._el.focus();
301             }, 0);
302         }
303         return this;
304     };
305 /**
306  * Todo generalise and formalise
307  * and normalise adding
308  * elements to the domUtil.
309  * elementsContainer
310  * @returns {Element}
311  * @private
312 */
313 this._createElement = function ()
314     {
315         var div = document.createElement
316             ('div');
317         div.setAttribute('
318             contentEditable', 'true');
319         div.className = Type.Settings.
320             prefix + 'input';
321         Type.DomUtilities.
322             getElementsContainer().
323             appendChild(div);
324         return div;
325     };
326 }}.call(Type.Input.prototype);
327 Type.Input.Filter = {};
328 module.exports = Type.Input;

```

Listing 18: input.js

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6   * Creates a new Type event
7   * @constructor
8   */
9  Type.Events.Type = function () {
10    this.canceled = false;
11  };
12
13  (function () {
14
15    /**
16     * Sets or gets data for this
17     * event. Parameters can be set
18     * and retrieved like in jQuery:
19     *
20     * Call data with no params to
21     * retrieve all data set:
22     * this.data() -> {}
23
24     * Pass a single string to get
25     * specific data:
26     * this.data('foo')
27
28     * Pass a name value combination
29     * to set data
30     * this.data('foo', 'bar')
31
32     * Pass an object to set multiple
33     * data
34     * this.data({foo: 'foo', 'bar': 'bar'})
35
36     * @param {(string|Object)} data -
37     *   Either a plain object
38     *   with keys and values to be
39     *   set or a string that will
40     *   be used as a name for a
41
42     data setting. If you pass a
43     string, pass a second
44     parameter to set that data
45     or no second parameter to
46     retrieve that data.
47     * @param {*} [value] - If the
48     *   first parameter is a string,
49     *   this value will be set to
50     *   the key of the given first
51     *   parameter. Any arbitrary
52     *   value can be set.
53     * @returns {Type.Events.Type
54     *   |{}|*} Returns this instance
55     *   if you set data or the
56     *   according value if you get
57     *   data. Will return all data
58     *   in an object of you pass
59     *   no parameters.
60     */
61     this.data = function (data, value)
62     {
63       // Initialize data object if not
64       // initialized yet
65       this._data = this._data || {};
66
67       // Pass a single option name to
68       // fetch it
69       if (typeof data === "string" &&
70         arguments.length === 1) {
71         return this._data[data];
72       }
73
74       // Pass an option name and a
75       // value to set it
76       if (typeof data === "string" &&
77         arguments.length === 2) {
78         data = {options: value};
79       }
80
81       // Pass an object of key-values
82       // to set them
83       if (typeof data === "object") {
84         Type.Utilities.extend(this,
85           _data, data);
86       }
87
88       // Data of no params have been
89       // passed, otherwise this for
90       // chaining
91       return arguments.length ? this :
92         this._data;
93     };
94
95    /**
96     * Sets this event instance to be
97     * cancelled
98     *
99     * @param {boolean} [doCancel] -
100     *   Set to false to uncancel
101     *   the event. All other values
102     *   or no value at all
103     *   will set the event to be
104     *   cancelled
105     * @returns {Type.Events.Type} -
106     *   This instance
107     */
108    this.cancel = function (doCancel)
109    {
110      this.canceled = doCancel !==
111        false;
112      return this;
113    };
114
115    }.call(Type.Events.Type.prototype);
116
117    module.exports = Type.Events.Type;
118  }
119
120  Listing 19: events/type.js

```

```

1  'use strict';
2
3  var Type = require('../core');
4
5  /**
6   * Creates a new Type input event.
7   * This is an abstraction for
8   * browser events that lead to an
9   * input in
10  * the editor.
11
12  * @param {Object} options - Object
13  * holding parameters for the
14  * event
15  * @param {string} [options.key] - A
16  * descriptive name for the key
17  * pressed as set in {@link Type
18  * .Events.Input.keyDownNames}.
19  * @param {number} [options.keyCode]
20  * - The key code of the key
21  * pressed
22  * @param {boolean} [options.shift]
23  * - Whether or not the shift key
24  * has
25  * been pressed together with
26  * the given key.
27  * @param {boolean} [options.alt] -
28  * Whether or not the alt key has
29  * been pressed together with
30  * the given key.
31  * @param {boolean} [options.ctrl] -
32  * Whether or not the control
33  * key has
34  * been pressed together with
35  * the given key.
36  * @param {boolean} [options.meta] -
37  * Whether or not the command
38  * key has
39  * been pressed together with
40  * the given key (for os x users)
41
42  * @constructor
43  */
44  Type.Events.Input = function (
45    options) {
46
47    options = options || {};
48
49    var charCode = (typeof e.which ===
50      "number" ? e.which : e.
51      keyCode,
52    options = {
53      key : Type.Events.Input.
54        charCode,
55      keyCode : charCode,
56      shift : e.shiftKey,
57      alt : e.altKey,
58      ctrl : e.ctrlKey,
59      meta : e.metaKey
60    };
61
62    return new Type.Events.Input (
63      options);
64
65  };
66
67  module.exports = Type.Events.Input;
68
69  Listing 20: events/input.js

```

```

1 'use strict';
2
3 var Input = require('..input');
4
5 /**
6  * Creates a caret filter. Will
7  * catch arrow key inputs,
8  * move the editor's caret and
9  * cancel the event.
10
11 * @param {Type} type
12 * @constructor
13
14 Input.Filter.Caret = function (type)
15 {
16   this._caret = type.getCaret();
17 };
18
19 (function () {
20   this.keys = {
21     left : 'moveLeft',
22     up : 'moveUp',
23     right : 'moveRight',
24     down : 'moveDown',
25
26     this.keys = {
27       left : 'moveLeft',
28       up : 'moveUp',
29       right : 'moveRight',
30       down : 'moveDown',
31
32       left : 'moveLeft',
33       up : 'moveUp',
34       right : 'moveRight',
35       down : 'moveDown',
36
37       left : 'moveLeft',
38       up : 'moveUp',
39       right : 'moveRight',
40       down : 'moveDown',
41
42       left : 'moveLeft',
43       up : 'moveUp',
44       right : 'moveRight',
45       down : 'moveDown',
46
47       left : 'moveLeft',
48       up : 'moveUp',
49       right : 'moveRight',
50       down : 'moveDown',
51
52       left : 'moveLeft',
53       up : 'moveUp',
54       right : 'moveRight',
55       down : 'moveDown',
56
57       left : 'moveLeft',
58       up : 'moveUp',
59       right : 'moveRight',
60       down : 'moveDown',
61
62       left : 'moveLeft',
63       up : 'moveUp',
64       right : 'moveRight',
65       down : 'moveDown',
66
67       left : 'moveLeft',
68       up : 'moveUp',
69       right : 'moveRight',
70       down : 'moveDown',
71
72       left : 'moveLeft',
73       up : 'moveUp',
74       right : 'moveRight',
75       down : 'moveDown',
76
77       left : 'moveLeft',
78       up : 'moveUp',
79       right : 'moveRight',
80       down : 'moveDown',
81
82       left : 'moveLeft',
83       up : 'moveUp',
84       right : 'moveRight',
85       down : 'moveDown',
86
87       left : 'moveLeft',
88       up : 'moveUp',
89       right : 'moveRight',
90       down : 'moveDown',
91
92       left : 'moveLeft',
93       up : 'moveUp',
94       right : 'moveRight',
95       down : 'moveDown',
96
97       left : 'moveLeft',
98       up : 'moveUp',
99       right : 'moveRight',
100      down : 'moveDown',
101
102      left : 'moveLeft',
103      up : 'moveUp',
104      right : 'moveRight',
105      down : 'moveDown',
106
107      left : 'moveLeft',
108      up : 'moveUp',
109      right : 'moveRight',
110      down : 'moveDown',
111
112      left : 'moveLeft',
113      up : 'moveUp',
114      right : 'moveRight',
115      down : 'moveDown',
116
117      left : 'moveLeft',
118      up : 'moveUp',
119      right : 'moveRight',
120      down : 'moveDown',
121
122      left : 'moveLeft',
123      up : 'moveUp',
124      right : 'moveRight',
125      down : 'moveDown',
126
127      left : 'moveLeft',
128      up : 'moveUp',
129      right : 'moveRight',
130      down : 'moveDown',
131
132      left : 'moveLeft',
133      up : 'moveUp',
134      right : 'moveRight',
135      down : 'moveDown',
136
137      left : 'moveLeft',
138      up : 'moveUp',
139      right : 'moveRight',
140      down : 'moveDown',
141
142      left : 'moveLeft',
143      up : 'moveUp',
144      right : 'moveRight',
145      down : 'moveDown',
146
147      left : 'moveLeft',
148      up : 'moveUp',
149      right : 'moveRight',
150      down : 'moveDown',
151
152      left : 'moveLeft',
153      up : 'moveUp',
154      right : 'moveRight',
155      down : 'moveDown',
156
157      left : 'moveLeft',
158      up : 'moveUp',
159      right : 'moveRight',
160      down : 'moveDown',
161
162      left : 'moveLeft',
163      up : 'moveUp',
164      right : 'moveRight',
165      down : 'moveDown',
166
167      left : 'moveLeft',
168      up : 'moveUp',
169      right : 'moveRight',
170      down : 'moveDown',
171
172      left : 'moveLeft',
173      up : 'moveUp',
174      right : 'moveRight',
175      down : 'moveDown',
176
177      left : 'moveLeft',
178      up : 'moveUp',
179      right : 'moveRight',
180      down : 'moveDown',
181
182      left : 'moveLeft',
183      up : 'moveUp',
184      right : 'moveRight',
185      down : 'moveDown',
186
187      left : 'moveLeft',
188      up : 'moveUp',
189      right : 'moveRight',
190      down : 'moveDown',
191
192      left : 'moveLeft',
193      up : 'moveUp',
194      right : 'moveRight',
195      down : 'moveDown',
196
197      left : 'moveLeft',
198      up : 'moveUp',
199      right : 'moveRight',
200      down : 'moveDown',
201
202      left : 'moveLeft',
203      up : 'moveUp',
204      right : 'moveRight',
205      down : 'moveDown',
206
207      left : 'moveLeft',
208      up : 'moveUp',
209      right : 'moveRight',
210      down : 'moveDown',
211
212      left : 'moveLeft',
213      up : 'moveUp',
214      right : 'moveRight',
215      down : 'moveDown',
216
217      left : 'moveLeft',
218      up : 'moveUp',
219      right : 'moveRight',
220      down : 'moveDown',
221
222      left : 'moveLeft',
223      up : 'moveUp',
224      right : 'moveRight',
225      down : 'moveDown',
226
227      left : 'moveLeft',
228      up : 'moveUp',
229      right : 'moveRight',
230      down : 'moveDown',
231
232      left : 'moveLeft',
233      up : 'moveUp',
234      right : 'moveRight',
235      down : 'moveDown',
236
237      left : 'moveLeft',
238      up : 'moveUp',
239      right : 'moveRight',
240      down : 'moveDown',
241
242      left : 'moveLeft',
243      up : 'moveUp',
244      right : 'moveRight',
245      down : 'moveDown',
246
247      left : 'moveLeft',
248      up : 'moveUp',
249      right : 'moveRight',
250      down : 'moveDown',
251
252      left : 'moveLeft',
253      up : 'moveUp',
254      right : 'moveRight',
255      down : 'moveDown',
256
257      left : 'moveLeft',
258      up : 'moveUp',
259      right : 'moveRight',
260      down : 'moveDown',
261
262      left : 'moveLeft',
263      up : 'moveUp',
264      right : 'moveRight',
265      down : 'moveDown',
266
267      left : 'moveLeft',
268      up : 'moveUp',
269      right : 'moveRight',
270      down : 'moveDown',
271
272      left : 'moveLeft',
273      up : 'moveUp',
274      right : 'moveRight',
275      down : 'moveDown',
276
277      left : 'moveLeft',
278      up : 'moveUp',
279      right : 'moveRight',
280      down : 'moveDown',
281
282      left : 'moveLeft',
283      up : 'moveUp',
284      right : 'moveRight',
285      down : 'moveDown',
286
287      left : 'moveLeft',
288      up : 'moveUp',
289      right : 'moveRight',
290      down : 'moveDown',
291
292      left : 'moveLeft',
293      up : 'moveUp',
294      right : 'moveRight',
295      down : 'moveDown',
296
297      left : 'moveLeft',
298      up : 'moveUp',
299      right : 'moveRight',
300      down : 'moveDown',
301
302      left : 'moveLeft',
303      up : 'moveUp',
304      right : 'moveRight',
305      down : 'moveDown',
306
307      left : 'moveLeft',
308      up : 'moveUp',
309      right : 'moveRight',
310      down : 'moveDown',
311
312      left : 'moveLeft',
313      up : 'moveUp',
314      right : 'moveRight',
315      down : 'moveDown',
316
317      left : 'moveLeft',
318      up : 'moveUp',
319      right : 'moveRight',
320      down : 'moveDown',
321
322      left : 'moveLeft',
323      up : 'moveUp',
324      right : 'moveRight',
325      down : 'moveDown',
326
327      left : 'moveLeft',
328      up : 'moveUp',
329      right : 'moveRight',
330      down : 'moveDown',
331
332      left : 'moveLeft',
333      up : 'moveUp',
334      right : 'moveRight',
335      down : 'moveDown',
336
337      left : 'moveLeft',
338      up : 'moveUp',
339      right : 'moveRight',
340      down : 'moveDown',
341
342      left : 'moveLeft',
343      up : 'moveUp',
344      right : 'moveRight',
345      down : 'moveDown',
346
347      left : 'moveLeft',
348      up : 'moveUp',
349      right : 'moveRight',
350      down : 'moveDown',
351
352      left : 'moveLeft',
353      up : 'moveUp',
354      right : 'moveRight',
355      down : 'moveDown',
356
357      left : 'moveLeft',
358      up : 'moveUp',
359      right : 'moveRight',
360      down : 'moveDown',
361
362      left : 'moveLeft',
363      up : 'moveUp',
364      right : 'moveRight',
365      down : 'moveDown',
366
367      left : 'moveLeft',
368      up : 'moveUp',
369      right : 'moveRight',
370      down : 'moveDown',
371
372      left : 'moveLeft',
373      up : 'moveUp',
374      right : 'moveRight',
375      down : 'moveDown',
376
377      left : 'moveLeft',
378      up : 'moveUp',
379      right : 'moveRight',
380      down : 'moveDown',
381
382      left : 'moveLeft',
383      up : 'moveUp',
384      right : 'moveRight',
385      down : 'moveDown',
386
387      left : 'moveLeft',
388      up : 'moveUp',
389      right : 'moveRight',
390      down : 'moveDown',
391
392      left : 'moveLeft',
393      up : 'moveUp',
394      right : 'moveRight',
395      down : 'moveDown',
396
397      left : 'moveLeft',
398      up : 'moveUp',
399      right : 'moveRight',
400      down : 'moveDown',
401
402      left : 'moveLeft',
403      up : 'moveUp',
404      right : 'moveRight',
405      down : 'moveDown',
406
407      left : 'moveLeft',
408      up : 'moveUp',
409      right : 'moveRight',
410      down : 'moveDown',
411
412      left : 'moveLeft',
413      up : 'moveUp',
414      right : 'moveRight',
415      down : 'moveDown',
416
417      left : 'moveLeft',
418      up : 'moveUp',
419      right : 'moveRight',
420      down : 'moveDown',
421
422      left : 'moveLeft',
423      up : 'moveUp',
424      right : 'moveRight',
425      down : 'moveDown',
426
427      left : 'moveLeft',
428      up : 'moveUp',
429      right : 'moveRight',
430      down : 'moveDown',
431
432      left : 'moveLeft',
433      up : 'moveUp',
434      right : 'moveRight',
435      down : 'moveDown',
436
437      left : 'moveLeft',
438      up : 'moveUp
```



```

1  'use strict';
2
3  var Input = require('..input');
4
5  /**
6   * Creates a command filter. Will
7   * text formatting keyboard
8   * shortcuts and call
9   * the according formatting methods.
10  *
11  * todo should listen for key codes
12  * and not keys
13  *
14  * @param type
15  * @param {Type.Input} input
16  * @constructor
17  */
18  Input.Filter.Command = function (
19    type, input) {
20    this._selection = type.
21      getSelection();
22    //this._formatting = type.
23      getFormatter();
24    this._content = input.getContent()
25  };
26
27  (function () {
28    this.keys = {
29      66 : 'command', // b
30      73 : 'command', // i
31      83 : 'command', // s
32      85 : 'command', // u
33    };
34
35    this.tags = {
36      66 : 'strong',
37      73 : 'em',
38      83 : 's',
39      85 : 'u',
40    };
41
42    /**
43     * todo format stuff when nothing
44     * is selected
45     * @param {Type.Events.Input} e
46     */
47    this.command = function (e) {
48      var sel;
49
50      if (e.cmd) {
51        sel = this._selection.save();
52        this._content.format(this.tags
53          [e.key], this._selection.
54            getRange());
55        this._selection.restore(sel);
56        e.cancel();
57      }
58    };
59
60    this.call(Input.Filter.Command.
61      prototype);
62
63    module.exports = Input.Filter.
64      Command;
65  })();
66
67 Listing 23:
68 input_filters/command.js
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

41 newOffset = this._caret.
42   getOffset() + moveChars;
43 } else {
44   range = this._selection.
45     getRange();
46   newOffset = range.
47     getStartOffset(this._root
48     );
49   this._selection.unselect();
50 }
51
52
53
54
55   this._caret._blink();
56 }
57 }.call(Input.Filter.Remove,
58   prototype);
59 module.exports = Input.Filter.Remove;

```

Listing 24: input_filters/remove.js

```

1  'use strict';
2
3  var Input = require('./input');
4
5  /**
6   * Creates a caret filter. Will
7   * catch arrow key inputs,
8   * move the editor's caret and
9   * cancel the event.
10  * @param {Type} type
11  * @param {Type.Input} [input]
12  * @constructor
13  Input.Filter.LineBreaks = function (
14    type, input) {
15    this._writer = type.getWriter();
16    this._caret = type.getCaret();
17
18    'use strict';
19
20    var Input = require('./input');
21
22    /**
23     * Creates a caret filter. Will
24     * catch arrow key inputs,
25     * move the editor's caret and
26     * cancel the event.
27     * @param {Type} type
28     * @param {Type.Input} [input]
29     * @constructor
30     Input.Filter.LineBreaks = function (
31       type, input) {
32         this._writer = type.getWriter();
33         this._caret = type.getCaret();
34
35         var br = document.createElement
36           ('br');
37         this._writer.insertHTML(this.
38           _caret.textNode, this.
39           _caret.offset, br);
40         this._caret.moveRight();
41         e.cancel();
42       };
43     }
44
45     }.call(Input.Filter.LineBreaks,
46       prototype);
47
48     module.exports = Input.Filter.
49       LineBreaks;

```

Listing 25: input_filters/line_breaks.js

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6   * @param {Type} type
7   * @constructor
8   Type.UndoManager = function (type) {
9
10
11
12
13   this._type = type;
14   this._stack = [];
15   this._pointer = 0;
16   this.lastActionReceived = null;
17   this.mergeDebounce = 500;
18
19   (function () {
20
21     * @param {Type.Actions.Type|Type.
22     *   Actions.Insert|*} action
23     * @returns {Type.UndoManager}
24     */
25     this.push = function (action) {
26       this._stack.length = this._stack
27         .length === 0 ? 0 : this.
28         _pointer + 1;
29       if (this.shouldBeMerged(action))

```

```

28         this._stack[this._pointer].
29         merge(action)
30     } else {
31         this._stack.push(action);
32         this._pointer = this._stack.
33         length - 1;
34     }
35     this.lastActionReceived = Date.
36     now();
37     return this;
38 };
39 /**
40  * @param action
41  * @returns {boolean}
42  * this.shouldBeMerged = function (
43  *   action) {
44  *     if (this.lastActionReceived ===
45  *       null) {
46  *       return false;
47  *     }
48     if (Date.now() > this.
49     lastActionReceived + this.
50     mergeDebounce) {
51       return false;
52     }
53     return !(this._stack.length &&
54     this._stack[this._pointer].
55     mergeable(action));
56 };
57 /**
58  * @param {*} [sourceId]
59  * @param {number} [steps]
60  * @returns {Type.UndoManager}
61  */
62 this.undo = function (sourceId,
63 steps) {
64     steps = steps === undefined ? 1
65     : steps;
66     //for (steps; steps > 0; steps
67     -= 1) {
68         // this._pointer++;
69         if (this._pointer < 0) {
70             this._pointer = -1;
71             break;
72         }
73         this._stack[this._pointer].
74         undo(this.
75         _getCharacterShift());
76         // this._pointer--;
77         while (steps > 0 && this.
78         _pointer > -1) {
79             if (this._stack[this._pointer
80             ].sourceId === sourceId
81             || sourceId === undefined
82             ) {
83                 this._stack[this._pointer].
84                 _undo(this.
85                 _getCharacterShift());
86                 steps -= 1;
87                 this._pointer -= 1;
88             }
89             return this;
90         }
91         //**
92         * @param {*} [sourceId]
93         * @param {number} [steps]
94         * @returns {Type.UndoManager}
95         */
96         this.redo = function (sourceId,
97         steps) {
98             var stackLen = this._stack.
99             length;
100             steps = steps === undefined ? 1
101             : steps;
102             //for (steps; steps > 0; steps
103             -= 1) {
104                 // if (this._pointer < 0) {
105                 //     this._pointer = -1;
106                 //     break;
107                 // }
108                 // this._stack[this._pointer].
109                 // _redo(this.
110                 // _getCharacterShift());
111                 // steps--;
112                 // return this;
113             }
114             //**
115             * Will iterate through the stack
116             * (beginning from its end)
117             * and collect all character
118             * insertions and removals and
119             * return them. This can be used

```

```

121      * bei actions to shift the
122      * their character offset to which
123      * they apply their changes.
124      * @param {number} [targetPointer]
125      *   - The stack pointer
126      *     to which all character
127      *     shifts shall be collected
128      * @returns {number[][]} - A map
129      *   of insertions and removals
130      *   First dimensions is at
131      *   which offsets characters
132      *   have
133      *   changed. Second dimension
134      *   is the number of characters
135      *   that have been added or
136      *   removed.
137      * @private
138      */
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156

```

```

this._getCharacterShift = function
(targetPointer) {
    targetPointer = targetPointer
    _pointer + 1 :
    targetPointer;
    var len = this._stack.length -
    1,
    shifts = [],
    shift,
    i, j;
    for (i = len; i >= targetPointer
    ; i -= 1) {
        shift = this._stack[i].

```

```

        getCharacterShift();
        for (j = 0; j < shift.length;
        j++) {
            shifts.push( shift[j] );
        }
        //shifts.concat(shift);
    }
    return shifts;
}
}).call(Type.UndoManager.prototype);
module.exports = Type.UndoManager;
Listing 26: undo_manager.js

```

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6   * Creates a new Content class
7   *
8   * This class can be used to
9   * manipulate the editor's
10  * contents and will make sure any
11  * action performed
12  * is undoable and re-doable.
13  *
14  * @param {Type} type
15  * @constructor
16  */
17  Type.Content = function (type) {
18      this._sourceId = this.
19      _createUniqueSourceId();
20      this._undoManager = type.
21      getUndoManager();
22      this._writer = type.getWriter();
23      this._formatter = type.
24
25
26
27
28
29
30
31
32
33
34
35
36

```

```

    getFormatter();
    this._root = type.getRoot();
    this._type = type;
};
(function () {
    /**
     * Inserts text to the editor's
     * contents and pushes an
     * action to the undo manager{}
     *
     * @param {Text|Number} textNode -
     *   The text node in which the
     *   contents should be inserted
     *
     * @param {Number|String} offset -
     *   The character offset in the
     *   text node at which the
     *   contents should be inserted
     *
     * @param {String} [content] - The
     *   text that should be
     *   inserted
     *
     * @returns {Type.Content} - This
     *   instance

```

```

37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

```

```

    */
    this.insert = function (textNode,
    offset, content) {
        // If only an offset and
        // contents were given
        if (arguments.length === 2) {
            var nodeInfo = Type.TextWalker
            .nodeAt(this._root,
            textNode);
            content = offset;
            offset = nodeInfo.offset;
            textNode = nodeInfo.node;
        }
        // Change contents
        this._writer.insertText(textNode
        , offset, content);
        // Undo capabilities
        var absOffset = Type.TextWalker.
        offset(this._root, textNode
        , 0, offset);

```

```

53   var insertion = new Type.Actions
54   .Insert(this._sourceId,
55   this._type, absOffset,
56   content);
57   this._undoManager.push(insertion
58   );
59   // Chaining
60   return this;
61
62   };
63   /**
64   * Removes the text inside a given
65   range from the contents
66   *
67   * @param {Type.Range|Number}
68   range - The text range that
69   should
70   be removed from the
71   contents. This parameter can
72   also be
73   the start offset
74   * @param {Number} numCharacters -
75   If this parameter is set
76   the
77   first parameter will be
78   interpreted as a number and
79   is the
80   start offset in the text.
81   This parameter will be the
82   number
83   of character to be removed
84   beginning from the start
85   offset.
86   * @returns {Type.Content} - This
87   instance
88   this.remove = function (range,
89   numCharacters) {
90   // If only an offset
91   numCharacters were given
92   if (arguments.length === 2) {
93   range = Type.Range.
94   fromPositions(this._root,
95   range, range +
96   numCharacters);
97   }
98   // Undo capabilities
99   var removal = Type.Actions.
100   Remove.fromRange(this.
101   _sourceId, this._type,
102   range);
103   this._undoManager.push(removal);
104   // Change contents
105   this._writer.remove(range);
106   // Chaining
107   return this;
108   };
109   /**
110   * Formats a given text range
111   *
112   * @param {String} tag - The HTML
113   tag the text should
114   be formatted with
115   * @param {Type.Range|number}
116   range - The range of text
117   that should be formatted or
118   a number that will be
119   formatting
120   * @param {number} [end] - If the
121   second parameter that
122   was given is a start offset
123   , this will be the end
124   offset in the text that
125   will be formatted.
126   * @returns {Type.Content} - This
127   instance
128   this.format = function (tag, range
129   , end) {
130   // If positions instead of a
131   range were given
132   if (arguments.length === 3) {
133   range = Type.Range.
134   fromPositions(this._root,
135   range, end);
136   }
137   // Change contents
138   var nodes = this._formatter.
139   format(tag, range);
140   this._undoManager.push(
141   formatting);
142   // Chaining
143   return this;
144   };
145   /**
146   * Formats a given text range
147   *
148   * @param {String} tag - The HTML
149   tag the text should
150   be formatted with
151   * @param {Type.Range|number}
152   range - The range of text
153   that should be formatted or
154   a number that will be
155   formatting
156   * @param {number} [end] - If the
157   second parameter that
158   was given is a start offset
159   , this will be the end
160   offset in the text that
161   will be formatted.
162   * @returns {Type.Content} - This
163   instance
164   this.removeFormat = function (tag,

```

```

138         range, end) {
139             // If positions instead of a
140             // range were given
141             if (arguments.length === 3) {
142                 range = Type.Range.
143                 fromPositions(this._root,
144                             range, end);
145             }
146             // Change contents
147             this._formatter.removeFormat(tag
148                                     , range);
149             // Chaining
150             return this;
151         };
152         /**
153         range, end) {
154             // If positions instead of a
155             // range were given
156             if (arguments.length === 3) {
157                 range = Type.Range.
158                 fromPositions(this._root,
159                             range, end);
160             }
161             // Change contents
162             this._formatter.removeFormat(tag
163                                     , range);
164             // Chaining
165             return this;
166         };
167         /**
168         range, end) {
169             // If positions instead of a
170             // range were given
171             if (arguments.length === 3) {
172                 range = Type.Range.
173                 fromPositions(this._root,
174                             range, end);
175             }
176             // Change contents
177             this._formatter.removeFormat(tag
178                                     , range);
179             // Chaining
180             return this;
181         };
182         /**

```

Listing 27: content.js

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6   * Creates a new Type action
7   * @param {*} sourceId - An
8   *   arbitrary key identifying the
9   *   author
10  * @param {boolean} [undone] - The
11  *   state of this action
12  * @constructor
13  Type.Actions.Type = function (
14    sourceId, undone) {
15    this.sourceId = sourceId;
16    this.undone = undone || false;
17  };
18  (function () {
19      /**
20       * Performs this action
21       * @returns {Type.Actions.Type} -
22       *   This instance
23       */
24      this.execute = function () {
25          this.undone = false;
26          return this;
27      };
28      /**
29       * Revokes this action
30       * @returns {Type.Actions.Type} -
31       *   This instance
32       */
33      this.undo = function () {
34          this.undone = true;
35          return this;
36      };
37  })();
38
39  /**
40   * Returns if a given action can
41   * be merged with this
42   * action
43   * @param {Type.Actions.Type|*}
44   *   that
45   * @returns {boolean}
46   */
47  this.mergeable = function (that) {
48      return false;
49  };
50  /**
51   * Merges a given action with this
52   * action
53   * @param {Type.Actions.Type|*}
54   *   that
55   * @returns {Type.Actions.Type} -
56   *   This instance

```

```

56 */
57 this.merge = function (that) {
58     return this;
59 };
60 /**
61  * Returns the offsets and number
62  * of characters
63  * this actions inserts or removes
64  *
65  * @returns {number[][]}
66  */
67 this.setCharacterShift = function
68     () {
69     return [[0,0]];
70 };
71 /**
72  * Calculates the number of
73     characters a given
74     offset must be adjusted based
75     on the given
76     character shifts.
77  *
78  * @param {number} offset - The
79     character offset
80     for which the number of
81     characters must
82     be added or removed to
83     account for the
84     given shifts
85  *
86  * @param {number[][]} shifts -
87     The character
88     shifts that must be
89     accounted for
90  *
91  * @returns {number} - The number
92     of characters
93     that an offset must be
94     added to or
95     removed from to account for
96     the given
97

```

```

85  * shifts.
86  * @private
87  */
88 this._getShiftTo = function (
89     offset, shifts) {
90     var adjustment = 0, len =
91     shifts.length, i;
92     for (i = 0; i < len; i += 1)
93     if (shifts[i][0] <= offset)
94     adjustment += shifts[i]
95     [1];
96     return adjustment;
97 };
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

Listing 28: actions/type.js

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6   * Creates a new Type action
7   * @param {*} sourceId - An
8   * arbitrary key identifying the
9   * author
10  *
11  * @param {Type} type - A type
12  * instance on which the action
13  * should be executed
14  *
15  * @param {Number} offset - The
16  * character offset at which the
17  * text should be inserted
18  *
19  * @param {String} text - The text
20  * containing HTML that
21  * should be inserted
22  *
23  * @param {boolean} [undone] - The
24  * state of this action
25  *
26  * @constructor
27
28  * Inherit from general Type action
29  */
30 Type.OOP.inherits(Type.Actions.
31     Insert, Type.Actions.Type);
32
33 (function () {
34     /**
35      * Inserts text in the editor
36      * @param {Number[][]} shifts
37      * @returns {Type.Actions.Insert}
38      * - This instance
39
40      * this.execute = function (shifts) {
41      *     var len = this._stack.length,
42      *     nodeInfo, i, adj;
43      *     for (i = 0; i < len; i += 1) {
44      *         adj = this._getShiftTo(this.
45      *             _stack[i].start, shifts);
46      *         nodeInfo = Type.TextWalker.
47      *             nodeAt(this._root, this.
48      *                 _stack[i].start + adj);
49      *         this._writer.insertText(
50      *             nodeInfo.node, nodeInfo.
51      *                 offset, this._stack[i].
52      *                     text);
53      *     }
54      *     this._caret.setOffset(this.
55      *         _stack[len-1].end + adj);
56

```

```

48 this.undone = false;
49 return this;
50 };
51
52 /**
53  * Revokes this action
54  * @param {Number[][]} shifts
55  * @returns {Type.Actions.Insert}
56  * - This instance
57  */
58 this.undo = function (shifts) {
59   var len = this._stack.length,
60   range, i, adj;
61   for (i = len - 1; i >= 0; i -= 1) {
62     adj = this._getShiftTo(this._stack[i].start, shifts);
63     range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
64     this._writer.remove(range);
65   }
66   this._caret.setOffset(this._stack[0].start + adj);
67   this.undone = true;
68   return this;
69 };
70
71 /**
72  * Returns if a given action can be merged with this action
73  * @param {*} that
74  * @returns {boolean}
75  */
76 this.mergeable = function (that) {
77   return that instanceof Type.Actions.Insert;
78 };
79
80 /**
81
82   * Merges a given action with this action
83   * @param {Type.Actions.Insert|*} that
84   * @returns {Type.Actions.Insert} - This instance
85   */
86 this.merge = function (that) {
87   var stack = that.getStack(),
88   length = stack.length,
89   i;
90   for (i = 0; i < length; i += 1) {
91     this.add(stack[i].start, stack[i].text);
92   }
93   return this;
94 };
95
96 /** Returns the offsets and number of characters
97  * this actions inserts
98  * @returns {number[][]}
99  */
100 this.getCharacterShift = function () {
101   var shifts, shift, len, stck, i;
102   if (this.undone) {
103     return [[0,0]];
104   }
105   shifts = [];
106   len = this._stack.length;
107   for (i = 0; i < len; i += 1) {
108     stck = this._stack[i];
109     shift = [stck.start, stck.end
110
111     this.undone = false;
112     return this;
113   }
114
115   /**
116    * Revokes this action
117    * @param {Number[][]} shifts
118    * @returns {Type.Actions.Insert}
119    * - This instance
120    */
121   this.undo = function (shifts) {
122     var len = this._stack.length,
123     range, i, adj;
124     for (i = len - 1; i >= 0; i -= 1) {
125       adj = this._getShiftTo(this._stack[i].start, shifts);
126       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
127       this._writer.remove(range);
128     }
129     this._caret.setOffset(this._stack[0].start + adj);
130     this.undone = true;
131     return this;
132   };
133
134   /** Returns if a given action can be merged with this action
135   * @param {*} that
136   * @returns {boolean}
137   */
138   this.mergeable = function (that) {
139     return that instanceof Type.Actions.Insert;
140   };
141
142   /**
143    * Merges a given action with this action
144    * @param {Type.Actions.Insert|*} that
145    * @returns {Type.Actions.Insert} - This instance
146    */
147   this.merge = function (that) {
148     var stack = that.getStack(),
149     length = stack.length,
150     i;
151     for (i = 0; i < length; i += 1) {
152       this.add(stack[i].start, stack[i].text);
153     }
154     return this;
155   };
156
157   /** Returns the offsets and number of characters
158   * this actions inserts
159   * @returns {number[][]}
160   */
161   this.getCharacterShift = function () {
162     var shifts, shift, len, stck, i;
163     if (this.undone) {
164       return [[0,0]];
165     }
166     shifts = [];
167     len = this._stack.length;
168     for (i = 0; i < len; i += 1) {
169       stck = this._stack[i];
170       shift = [stck.start, stck.end
171
172     this.undone = false;
173     return this;
174   }
175
176   /**
177    * Revokes this action
178    * @param {Number[][]} shifts
179    * @returns {Type.Actions.Insert}
180    * - This instance
181    */
182   this.undo = function (shifts) {
183     var len = this._stack.length,
184     range, i, adj;
185     for (i = len - 1; i >= 0; i -= 1) {
186       adj = this._getShiftTo(this._stack[i].start, shifts);
187       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
188       this._writer.remove(range);
189     }
190     this._caret.setOffset(this._stack[0].start + adj);
191     this.undone = true;
192     return this;
193   };
194
195   /** Returns if a given action can be merged with this action
196   * @param {*} that
197   * @returns {boolean}
198   */
199   this.mergeable = function (that) {
200     return that instanceof Type.Actions.Insert;
201   };
202
203   /**
204    * Merges a given action with this action
205    * @param {Type.Actions.Insert|*} that
206    * @returns {Type.Actions.Insert} - This instance
207    */
208   this.merge = function (that) {
209     var stack = that.getStack(),
210     length = stack.length,
211     i;
212     for (i = 0; i < length; i += 1) {
213       this.add(stack[i].start, stack[i].text);
214     }
215     return this;
216   };
217
218   /** Returns the offsets and number of characters
219   * this actions inserts
220   * @returns {number[][]}
221   */
222   this.getCharacterShift = function () {
223     var shifts, shift, len, stck, i;
224     if (this.undone) {
225       return [[0,0]];
226     }
227     shifts = [];
228     len = this._stack.length;
229     for (i = 0; i < len; i += 1) {
230       stck = this._stack[i];
231       shift = [stck.start, stck.end
232
233     this.undone = false;
234     return this;
235   }
236
237   /**
238    * Revokes this action
239    * @param {Number[][]} shifts
240    * @returns {Type.Actions.Insert}
241    * - This instance
242    */
243   this.undo = function (shifts) {
244     var len = this._stack.length,
245     range, i, adj;
246     for (i = len - 1; i >= 0; i -= 1) {
247       adj = this._getShiftTo(this._stack[i].start, shifts);
248       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
249       this._writer.remove(range);
250     }
251     this._caret.setOffset(this._stack[0].start + adj);
252     this.undone = true;
253     return this;
254   };
255
256   /** Returns if a given action can be merged with this action
257   * @param {*} that
258   * @returns {boolean}
259   */
260   this.mergeable = function (that) {
261     return that instanceof Type.Actions.Insert;
262   };
263
264   /**
265    * Merges a given action with this action
266    * @param {Type.Actions.Insert|*} that
267    * @returns {Type.Actions.Insert} - This instance
268    */
269   this.merge = function (that) {
270     var stack = that.getStack(),
271     length = stack.length,
272     i;
273     for (i = 0; i < length; i += 1) {
274       this.add(stack[i].start, stack[i].text);
275     }
276     return this;
277   };
278
279   /** Returns the offsets and number of characters
280   * this actions inserts
281   * @returns {number[][]}
282   */
283   this.getCharacterShift = function () {
284     var shifts, shift, len, stck, i;
285     if (this.undone) {
286       return [[0,0]];
287     }
288     shifts = [];
289     len = this._stack.length;
290     for (i = 0; i < len; i += 1) {
291       stck = this._stack[i];
292       shift = [stck.start, stck.end
293
294     this.undone = false;
295     return this;
296   }
297
298   /**
299    * Revokes this action
300    * @param {Number[][]} shifts
301    * @returns {Type.Actions.Insert}
302    * - This instance
303    */
304   this.undo = function (shifts) {
305     var len = this._stack.length,
306     range, i, adj;
307     for (i = len - 1; i >= 0; i -= 1) {
308       adj = this._getShiftTo(this._stack[i].start, shifts);
309       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
310       this._writer.remove(range);
311     }
312     this._caret.setOffset(this._stack[0].start + adj);
313     this.undone = true;
314     return this;
315   };
316
317   /** Returns if a given action can be merged with this action
318   * @param {*} that
319   * @returns {boolean}
320   */
321   this.mergeable = function (that) {
322     return that instanceof Type.Actions.Insert;
323   };
324
325   /**
326    * Merges a given action with this action
327    * @param {Type.Actions.Insert|*} that
328    * @returns {Type.Actions.Insert} - This instance
329    */
330   this.merge = function (that) {
331     var stack = that.getStack(),
332     length = stack.length,
333     i;
334     for (i = 0; i < length; i += 1) {
335       this.add(stack[i].start, stack[i].text);
336     }
337     return this;
338   };
339
340   /** Returns the offsets and number of characters
341   * this actions inserts
342   * @returns {number[][]}
343   */
344   this.getCharacterShift = function () {
345     var shifts, shift, len, stck, i;
346     if (this.undone) {
347       return [[0,0]];
348     }
349     shifts = [];
350     len = this._stack.length;
351     for (i = 0; i < len; i += 1) {
352       stck = this._stack[i];
353       shift = [stck.start, stck.end
354
355     this.undone = false;
356     return this;
357   }
358
359   /**
360    * Revokes this action
361    * @param {Number[][]} shifts
362    * @returns {Type.Actions.Insert}
363    * - This instance
364    */
365   this.undo = function (shifts) {
366     var len = this._stack.length,
367     range, i, adj;
368     for (i = len - 1; i >= 0; i -= 1) {
369       adj = this._getShiftTo(this._stack[i].start, shifts);
370       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
371       this._writer.remove(range);
372     }
373     this._caret.setOffset(this._stack[0].start + adj);
374     this.undone = true;
375     return this;
376   };
377
378   /** Returns if a given action can be merged with this action
379   * @param {*} that
380   * @returns {boolean}
381   */
382   this.mergeable = function (that) {
383     return that instanceof Type.Actions.Insert;
384   };
385
386   /**
387    * Merges a given action with this action
388    * @param {Type.Actions.Insert|*} that
389    * @returns {Type.Actions.Insert} - This instance
390    */
391   this.merge = function (that) {
392     var stack = that.getStack(),
393     length = stack.length,
394     i;
395     for (i = 0; i < length; i += 1) {
396       this.add(stack[i].start, stack[i].text);
397     }
398     return this;
399   };
400
401   /** Returns the offsets and number of characters
402   * this actions inserts
403   * @returns {number[][]}
404   */
405   this.getCharacterShift = function () {
406     var shifts, shift, len, stck, i;
407     if (this.undone) {
408       return [[0,0]];
409     }
410     shifts = [];
411     len = this._stack.length;
412     for (i = 0; i < len; i += 1) {
413       stck = this._stack[i];
414       shift = [stck.start, stck.end
415
416     this.undone = false;
417     return this;
418   }
419
420   /**
421    * Revokes this action
422    * @param {Number[][]} shifts
423    * @returns {Type.Actions.Insert}
424    * - This instance
425    */
426   this.undo = function (shifts) {
427     var len = this._stack.length,
428     range, i, adj;
429     for (i = len - 1; i >= 0; i -= 1) {
430       adj = this._getShiftTo(this._stack[i].start, shifts);
431       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
432       this._writer.remove(range);
433     }
434     this._caret.setOffset(this._stack[0].start + adj);
435     this.undone = true;
436     return this;
437   };
438
439   /** Returns if a given action can be merged with this action
440   * @param {*} that
441   * @returns {boolean}
442   */
443   this.mergeable = function (that) {
444     return that instanceof Type.Actions.Insert;
445   };
446
447   /**
448    * Merges a given action with this action
449    * @param {Type.Actions.Insert|*} that
450    * @returns {Type.Actions.Insert} - This instance
451    */
452   this.merge = function (that) {
453     var stack = that.getStack(),
454     length = stack.length,
455     i;
456     for (i = 0; i < length; i += 1) {
457       this.add(stack[i].start, stack[i].text);
458     }
459     return this;
460   };
461
462   /** Returns the offsets and number of characters
463   * this actions inserts
464   * @returns {number[][]}
465   */
466   this.getCharacterShift = function () {
467     var shifts, shift, len, stck, i;
468     if (this.undone) {
469       return [[0,0]];
470     }
471     shifts = [];
472     len = this._stack.length;
473     for (i = 0; i < len; i += 1) {
474       stck = this._stack[i];
475       shift = [stck.start, stck.end
476
477     this.undone = false;
478     return this;
479   }
480
481   /**
482    * Revokes this action
483    * @param {Number[][]} shifts
484    * @returns {Type.Actions.Insert}
485    * - This instance
486    */
487   this.undo = function (shifts) {
488     var len = this._stack.length,
489     range, i, adj;
490     for (i = len - 1; i >= 0; i -= 1) {
491       adj = this._getShiftTo(this._stack[i].start, shifts);
492       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
493       this._writer.remove(range);
494     }
495     this._caret.setOffset(this._stack[0].start + adj);
496     this.undone = true;
497     return this;
498   };
499
500   /** Returns if a given action can be merged with this action
501   * @param {*} that
502   * @returns {boolean}
503   */
504   this.mergeable = function (that) {
505     return that instanceof Type.Actions.Insert;
506   };
507
508   /**
509    * Merges a given action with this action
510    * @param {Type.Actions.Insert|*} that
511    * @returns {Type.Actions.Insert} - This instance
512    */
513   this.merge = function (that) {
514     var stack = that.getStack(),
515     length = stack.length,
516     i;
517     for (i = 0; i < length; i += 1) {
518       this.add(stack[i].start, stack[i].text);
519     }
520     return this;
521   };
522
523   /** Returns the offsets and number of characters
524   * this actions inserts
525   * @returns {number[][]}
526   */
527   this.getCharacterShift = function () {
528     var shifts, shift, len, stck, i;
529     if (this.undone) {
530       return [[0,0]];
531     }
532     shifts = [];
533     len = this._stack.length;
534     for (i = 0; i < len; i += 1) {
535       stck = this._stack[i];
536       shift = [stck.start, stck.end
537
538     this.undone = false;
539     return this;
540   }
541
542   /**
543    * Revokes this action
544    * @param {Number[][]} shifts
545    * @returns {Type.Actions.Insert}
546    * - This instance
547    */
548   this.undo = function (shifts) {
549     var len = this._stack.length,
550     range, i, adj;
551     for (i = len - 1; i >= 0; i -= 1) {
552       adj = this._getShiftTo(this._stack[i].start, shifts);
553       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
554       this._writer.remove(range);
555     }
556     this._caret.setOffset(this._stack[0].start + adj);
557     this.undone = true;
558     return this;
559   };
560
561   /** Returns if a given action can be merged with this action
562   * @param {*} that
563   * @returns {boolean}
564   */
565   this.mergeable = function (that) {
566     return that instanceof Type.Actions.Insert;
567   };
568
569   /**
570    * Merges a given action with this action
571    * @param {Type.Actions.Insert|*} that
572    * @returns {Type.Actions.Insert} - This instance
573    */
574   this.merge = function (that) {
575     var stack = that.getStack(),
576     length = stack.length,
577     i;
578     for (i = 0; i < length; i += 1) {
579       this.add(stack[i].start, stack[i].text);
580     }
581     return this;
582   };
583
584   /** Returns the offsets and number of characters
585   * this actions inserts
586   * @returns {number[][]}
587   */
588   this.getCharacterShift = function () {
589     var shifts, shift, len, stck, i;
590     if (this.undone) {
591       return [[0,0]];
592     }
593     shifts = [];
594     len = this._stack.length;
595     for (i = 0; i < len; i += 1) {
596       stck = this._stack[i];
597       shift = [stck.start, stck.end
598
599     this.undone = false;
600     return this;
601   }
602
603   /**
604    * Revokes this action
605    * @param {Number[][]} shifts
606    * @returns {Type.Actions.Insert}
607    * - This instance
608    */
609   this.undo = function (shifts) {
610     var len = this._stack.length,
611     range, i, adj;
612     for (i = len - 1; i >= 0; i -= 1) {
613       adj = this._getShiftTo(this._stack[i].start, shifts);
614       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
615       this._writer.remove(range);
616     }
617     this._caret.setOffset(this._stack[0].start + adj);
618     this.undone = true;
619     return this;
620   };
621
622   /** Returns if a given action can be merged with this action
623   * @param {*} that
624   * @returns {boolean}
625   */
626   this.mergeable = function (that) {
627     return that instanceof Type.Actions.Insert;
628   };
629
630   /**
631    * Merges a given action with this action
632    * @param {Type.Actions.Insert|*} that
633    * @returns {Type.Actions.Insert} - This instance
634    */
635   this.merge = function (that) {
636     var stack = that.getStack(),
637     length = stack.length,
638     i;
639     for (i = 0; i < length; i += 1) {
640       this.add(stack[i].start, stack[i].text);
641     }
642     return this;
643   };
644
645   /** Returns the offsets and number of characters
646   * this actions inserts
647   * @returns {number[][]}
648   */
649   this.getCharacterShift = function () {
650     var shifts, shift, len, stck, i;
651     if (this.undone) {
652       return [[0,0]];
653     }
654     shifts = [];
655     len = this._stack.length;
656     for (i = 0; i < len; i += 1) {
657       stck = this._stack[i];
658       shift = [stck.start, stck.end
659
660     this.undone = false;
661     return this;
662   }
663
664   /**
665    * Revokes this action
666    * @param {Number[][]} shifts
667    * @returns {Type.Actions.Insert}
668    * - This instance
669    */
670   this.undo = function (shifts) {
671     var len = this._stack.length,
672     range, i, adj;
673     for (i = len - 1; i >= 0; i -= 1) {
674       adj = this._getShiftTo(this._stack[i].start, shifts);
675       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
676       this._writer.remove(range);
677     }
678     this._caret.setOffset(this._stack[0].start + adj);
679     this.undone = true;
680     return this;
681   };
682
683   /** Returns if a given action can be merged with this action
684   * @param {*} that
685   * @returns {boolean}
686   */
687   this.mergeable = function (that) {
688     return that instanceof Type.Actions.Insert;
689   };
690
691   /**
692    * Merges a given action with this action
693    * @param {Type.Actions.Insert|*} that
694    * @returns {Type.Actions.Insert} - This instance
695    */
696   this.merge = function (that) {
697     var stack = that.getStack(),
698     length = stack.length,
699     i;
700     for (i = 0; i < length; i += 1) {
701       this.add(stack[i].start, stack[i].text);
702     }
703     return this;
704   };
705
706   /** Returns the offsets and number of characters
707   * this actions inserts
708   * @returns {number[][]}
709   */
710   this.getCharacterShift = function () {
711     var shifts, shift, len, stck, i;
712     if (this.undone) {
713       return [[0,0]];
714     }
715     shifts = [];
716     len = this._stack.length;
717     for (i = 0; i < len; i += 1) {
718       stck = this._stack[i];
719       shift = [stck.start, stck.end
720
721     this.undone = false;
722     return this;
723   }
724
725   /**
726    * Revokes this action
727    * @param {Number[][]} shifts
728    * @returns {Type.Actions.Insert}
729    * - This instance
730    */
731   this.undo = function (shifts) {
732     var len = this._stack.length,
733     range, i, adj;
734     for (i = len - 1; i >= 0; i -= 1) {
735       adj = this._getShiftTo(this._stack[i].start, shifts);
736       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
737       this._writer.remove(range);
738     }
739     this._caret.setOffset(this._stack[0].start + adj);
740     this.undone = true;
741     return this;
742   };
743
744   /** Returns if a given action can be merged with this action
745   * @param {*} that
746   * @returns {boolean}
747   */
748   this.mergeable = function (that) {
749     return that instanceof Type.Actions.Insert;
750   };
751
752   /**
753    * Merges a given action with this action
754    * @param {Type.Actions.Insert|*} that
755    * @returns {Type.Actions.Insert} - This instance
756    */
757   this.merge = function (that) {
758     var stack = that.getStack(),
759     length = stack.length,
760     i;
761     for (i = 0; i < length; i += 1) {
762       this.add(stack[i].start, stack[i].text);
763     }
764     return this;
765   };
766
767   /** Returns the offsets and number of characters
768   * this actions inserts
769   * @returns {number[][]}
770   */
771   this.getCharacterShift = function () {
772     var shifts, shift, len, stck, i;
773     if (this.undone) {
774       return [[0,0]];
775     }
776     shifts = [];
777     len = this._stack.length;
778     for (i = 0; i < len; i += 1) {
779       stck = this._stack[i];
780       shift = [stck.start, stck.end
781
782     this.undone = false;
783     return this;
784   }
785
786   /**
787    * Revokes this action
788    * @param {Number[][]} shifts
789    * @returns {Type.Actions.Insert}
790    * - This instance
791    */
792   this.undo = function (shifts) {
793     var len = this._stack.length,
794     range, i, adj;
795     for (i = len - 1; i >= 0; i -= 1) {
796       adj = this._getShiftTo(this._stack[i].start, shifts);
797       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
798       this._writer.remove(range);
799     }
800     this._caret.setOffset(this._stack[0].start + adj);
801     this.undone = true;
802     return this;
803   };
804
805   /** Returns if a given action can be merged with this action
806   * @param {*} that
807   * @returns {boolean}
808   */
809   this.mergeable = function (that) {
810     return that instanceof Type.Actions.Insert;
811   };
812
813   /**
814    * Merges a given action with this action
815    * @param {Type.Actions.Insert|*} that
816    * @returns {Type.Actions.Insert} - This instance
817    */
818   this.merge = function (that) {
819     var stack = that.getStack(),
820     length = stack.length,
821     i;
822     for (i = 0; i < length; i += 1) {
823       this.add(stack[i].start, stack[i].text);
824     }
825     return this;
826   };
827
828   /** Returns the offsets and number of characters
829   * this actions inserts
830   * @returns {number[][]}
831   */
832   this.getCharacterShift = function () {
833     var shifts, shift, len, stck, i;
834     if (this.undone) {
835       return [[0,0]];
836     }
837     shifts = [];
838     len = this._stack.length;
839     for (i = 0; i < len; i += 1) {
840       stck = this._stack[i];
841       shift = [stck.start, stck.end
842
843     this.undone = false;
844     return this;
845   }
846
847   /**
848    * Revokes this action
849    * @param {Number[][]} shifts
850    * @returns {Type.Actions.Insert}
851    * - This instance
852    */
853   this.undo = function (shifts) {
854     var len = this._stack.length,
855     range, i, adj;
856     for (i = len - 1; i >= 0; i -= 1) {
857       adj = this._getShiftTo(this._stack[i].start, shifts);
858       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
859       this._writer.remove(range);
860     }
861     this._caret.setOffset(this._stack[0].start + adj);
862     this.undone = true;
863     return this;
864   };
865
866   /** Returns if a given action can be merged with this action
867   * @param {*} that
868   * @returns {boolean}
869   */
870   this.mergeable = function (that) {
871     return that instanceof Type.Actions.Insert;
872   };
873
874   /**
875    * Merges a given action with this action
876    * @param {Type.Actions.Insert|*} that
877    * @returns {Type.Actions.Insert} - This instance
878    */
879   this.merge = function (that) {
880     var stack = that.getStack(),
881     length = stack.length,
882     i;
883     for (i = 0; i < length; i += 1) {
884       this.add(stack[i].start, stack[i].text);
885     }
886     return this;
887   };
888
889   /** Returns the offsets and number of characters
890   * this actions inserts
891   * @returns {number[][]}
892   */
893   this.getCharacterShift = function () {
894     var shifts, shift, len, stck, i;
895     if (this.undone) {
896       return [[0,0]];
897     }
898     shifts = [];
899     len = this._stack.length;
900     for (i = 0; i < len; i += 1) {
901       stck = this._stack[i];
902       shift = [stck.start, stck.end
903
904     this.undone = false;
905     return this;
906   }
907
908   /**
909    * Revokes this action
910    * @param {Number[][]} shifts
911    * @returns {Type.Actions.Insert}
912    * - This instance
913    */
914   this.undo = function (shifts) {
915     var len = this._stack.length,
916     range, i, adj;
917     for (i = len - 1; i >= 0; i -= 1) {
918       adj = this._getShiftTo(this._stack[i].start, shifts);
919       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
920       this._writer.remove(range);
921     }
922     this._caret.setOffset(this._stack[0].start + adj);
923     this.undone = true;
924     return this;
925   };
926
927   /** Returns if a given action can be merged with this action
928   * @param {*} that
929   * @returns {boolean}
930   */
931   this.mergeable = function (that) {
932     return that instanceof Type.Actions.Insert;
933   };
934
935   /**
936    * Merges a given action with this action
937    * @param {Type.Actions.Insert|*} that
938    * @returns {Type.Actions.Insert} - This instance
939    */
940   this.merge = function (that) {
941     var stack = that.getStack(),
942     length = stack.length,
943     i;
944     for (i = 0; i < length; i += 1) {
945       this.add(stack[i].start, stack[i].text);
946     }
947     return this;
948   };
949
950   /** Returns the offsets and number of characters
951   * this actions inserts
952   * @returns {number[][]}
953   */
954   this.getCharacterShift = function () {
955     var shifts, shift, len, stck, i;
956     if (this.undone) {
957       return [[0,0]];
958     }
959     shifts = [];
960     len = this._stack.length;
961     for (i = 0; i < len; i += 1) {
962       stck = this._stack[i];
963       shift = [stck.start, stck.end
964
965     this.undone = false;
966     return this;
967   }
968
969   /**
970    * Revokes this action
971    * @param {Number[][]} shifts
972    * @returns {Type.Actions.Insert}
973    * - This instance
974    */
975   this.undo = function (shifts) {
976     var len = this._stack.length,
977     range, i, adj;
978     for (i = len - 1; i >= 0; i -= 1) {
979       adj = this._getShiftTo(this._stack[i].start, shifts);
980       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
981       this._writer.remove(range);
982     }
983     this._caret.setOffset(this._stack[0].start + adj);
984     this.undone = true;
985     return this;
986   };
987
988   /** Returns if a given action can be merged with this action
989   * @param {*} that
990   * @returns {boolean}
991   */
992   this.mergeable = function (that) {
993     return that instanceof Type.Actions.Insert;
994   };
995
996   /**
997    * Merges a given action with this action
998    * @param {Type.Actions.Insert|*} that
999    * @returns {Type.Actions.Insert} - This instance
1000    */
1001   this.merge = function (that) {
1002     var stack = that.getStack(),
1003     length = stack.length,
1004     i;
1005     for (i = 0; i < length; i += 1) {
1006       this.add(stack[i].start, stack[i].text);
1007     }
1008     return this;
1009   };
1010
1011   /** Returns the offsets and number of characters
1012   * this actions inserts
1013   * @returns {number[][]}
1014   */
1015   this.getCharacterShift = function () {
1016     var shifts, shift, len, stck, i;
1017     if (this.undone) {
1018       return [[0,0]];
1019     }
1020     shifts = [];
1021     len = this._stack.length;
1022     for (i = 0; i < len; i += 1) {
1023       stck = this._stack[i];
1024       shift = [stck.start, stck.end
1025
1026     this.undone = false;
1027     return this;
1028   }
1029
1030   /**
1031    * Revokes this action
1032    * @param {Number[][]} shifts
1033    * @returns {Type.Actions.Insert}
1034    * - This instance
1035    */
1036   this.undo = function (shifts) {
1037     var len = this._stack.length,
1038     range, i, adj;
1039     for (i = len - 1; i >= 0; i -= 1) {
1040       adj = this._getShiftTo(this._stack[i].start, shifts);
1041       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
1042       this._writer.remove(range);
1043     }
1044     this._caret.setOffset(this._stack[0].start + adj);
1045     this.undone = true;
1046     return this;
1047   };
1048
1049   /** Returns if a given action can be merged with this action
1050   * @param {*} that
1051   * @returns {boolean}
1052   */
1053   this.mergeable = function (that) {
1054     return that instanceof Type.Actions.Insert;
1055   };
1056
1057   /**
1058    * Merges a given action with this action
1059    * @param {Type.Actions.Insert|*} that
1060    * @returns {Type.Actions.Insert} - This instance
1061    */
1062   this.merge = function (that) {
1063     var stack = that.getStack(),
1064     length = stack.length,
1065     i;
1066     for (i = 0; i < length; i += 1) {
1067       this.add(stack[i].start, stack[i].text);
1068     }
1069     return this;
1070   };
1071
1072   /** Returns the offsets and number of characters
1073   * this actions inserts
1074   * @returns {number[][]}
1075   */
1076   this.getCharacterShift = function () {
1077     var shifts, shift, len, stck, i;
1078     if (this.undone) {
1079       return [[0,0]];
1080     }
1081     shifts = [];
1082     len = this._stack.length;
1083     for (i = 0; i < len; i += 1) {
1084       stck = this._stack[i];
1085       shift = [stck.start, stck.end
1086
1087     this.undone = false;
1088     return this;
1089   }
1090
1091   /**
1092    * Revokes this action
1093    * @param {Number[][]} shifts
1094    * @returns {Type.Actions.Insert}
1095    * - This instance
1096    */
1097   this.undo = function (shifts) {
1098     var len = this._stack.length,
1099     range, i, adj;
1100     for (i = len - 1; i >= 0; i -= 1) {
1101       adj = this._getShiftTo(this._stack[i].start, shifts);
1102       range = Type.Range.fromPositions(this._root, this._stack[i].start+adj; this._stack[i].end+adj);
1103       this._writer.remove(range);
1104     }
1105     this._caret.setOffset(this._stack[0].start + adj);
1106     this.undone = true;
1107     return this;
1108   };
1109
1110   /** Returns if a given action can be merged with this action
1111   * @param {*} that
1112   * @returns {boolean}
1113   */
1114   this.mergeable = function (that) {
1115     return that instanceof Type.Actions.Insert;
1116   };
1117
1118   /**
1119    * Merges a given action with this action
1120    * @param {Type.Actions
```



```

155     this._stack.splice(i, 0, {
156       start: start, end: end,
157       text: text});
158     break;
159   }
160   // Add to insertion if it
161   // overlaps with another
162   // insertion
163   if (start >= this._stack[i].
164       start && start <= this.
165       _stack[i].end) {
166     stackText = this._stack[i].
167     text;
168     insertPosition = start -
169     this._stack[i].start;
170     this._stack[i].text =
171     stackText.substr(0,
172     insertPosition) + text
173     + stackText.substr(
174     insertPosition);
175     this._stack[i].end += length
176     ;
177   }
178 }
179
180     break;
181   }
182   // Add to end
183   if (i+1 >= this._stack.length)
184   {
185     this._stack.push({start:
186     start, end: end, text:
187     text});
188     break;
189   }
190   // Insert between other
191   // insertions
192   if (this._stack[i].end < start
193       && (this._stack[i+1].
194       start < end)) {
195     this._stack.splice(i, 0, {
196       start: start, end: end,
197       text: text});
198     break;
199   }

```

Listing 29: actions/insert.js

```

1  'use strict';
2
3  var Type = require('../core');
4
5  /**
6   * Creates a new Type action
7   * @param {*} sourceId - An
8   *   arbitrary key identifying the
9   *   author
10  * of this action
11  * @param {Type} type - A type
12  *   instance on which the action
13  *   should be executed
14  * @param {Number} start - The
15  *   character offset from which
16  *   the
17  *   text should be removed
18  * @param {Number} end - The
19  *   character offset to which the
20
21   text should be removed
22   state of this action
23   @constructor
24   Type.Actions.Remove = function (
25     sourceId, type, start, end,
26     undone) {
27     this.sourceId = sourceId;
28     this.undone = undone || false;
29     this._writer = type.getWriter();
30     this._caret = type.getCaret();
31     this._root = type.getRoot();
32     this.start = start;
33     this.end = end;
34     this._contents = this._getContents
35     ();
36   };
37
38   * Inherit from general Type action
39   * @param {Number}[] shifts
40   * @returns {Type.Actions.Remove}
41   *   - This instance
42   *   this.execute = function (shifts) {
43     var adj = this._getShiftTo(this.
44     start, shifts),

```

```

43     range = Type.Range.
44     fromPositions(this._root,
45     this.start + adj, this.
46     end + adj);
47     this._writer.remove(range);
48     this._caret.setOffset(this.start
49     + adj);
50     this.undone = false;
51     return this;
52 };
53
54 /**
55  * Inserts the removed text again
56  * @param {Number[][]} shifts
57  * @returns {Type.Actions.Remove}
58  * - This instance
59 */
60 this.undo = function (shifts) {
61     var adj = this._getShiftTo(this.
62     start, shifts),
63     nodeInfo = Type.TextWalker.
64     nodeAt(this._root, this.
65     start + adj);
66     this._writer.insertHTML(nodeInfo
67     .node, nodeInfo.offset,
68     this._contents);
69     this._caret.setOffset(this.end +
70     adj);
71     this.undone = true;
72     return this;
73 };
74
75 /**
76  * Returns if a given action can
77  * be merged with this
78  * action
79  * @param {*} that
80  * @returns {boolean}
81 */
82 this.mergeable = function (that) {
83     return false; // Deactivated
84 }
85
86 range = Type.Range.
87 fromPositions(this._root,
88 this.start + adj, this.
89 end + adj);
90 this._writer.remove(range);
91 this._caret.setOffset(this.start
92 + adj);
93 this.undone = false;
94 return this;
95 };
96
97 /**
98  * Inserts the removed text again
99  * @param {Number[][]} shifts
100  * @returns {Type.Actions.Remove}
101  * - This instance
102 */
103 this.undo = function (shifts) {
104     var adj = this._getShiftTo(this.
105     start, shifts),
106     nodeInfo = Type.TextWalker.
107     nodeAt(this._root, this.
108     start + adj);
109     this._writer.insertHTML(nodeInfo
110     .node, nodeInfo.offset,
111     this._contents);
112     this._caret.setOffset(this.end +
113     adj);
114     this.undone = true;
115     return this;
116 };
117
118 /**
119  * Returns if a given action can
120  * be merged with this
121  * action
122  * @param {*} that
123  * @returns {boolean}
124 */
125 this.mergeable = function (that) {
126     return false; // Deactivated
127 }
128
129 range = Type.Range.
130 fromPositions(this._root,
131 this.start + adj, this.
132 end + adj);
133 this._writer.remove(range);
134 this._caret.setOffset(this.start
135 + adj);
136 this.undone = false;
137 return this;
138 };
139
140 /**
141  * Inserts the removed text again
142  * @param {Number[][]} shifts
143  * @returns {Type.Actions.Remove}
144  * - This instance
145 */
146 this.undo = function (shifts) {
147     var adj = this._getShiftTo(this.
148     start, shifts),
149     nodeInfo = Type.TextWalker.
150     nodeAt(this._root, this.
151     start + adj);
152     this._writer.insertHTML(nodeInfo
153     .node, nodeInfo.offset,
154     this._contents);
155     this._caret.setOffset(this.end +
156     adj);
157     this.undone = true;
158     return this;
159 };
160
161 /**
162  * Returns if a given action can
163  * be merged with this
164  * action
165  * @param {*} that
166  * @returns {boolean}
167 */
168 this.mergeable = function (that) {
169     return false; // Deactivated
170 }
171
172 range = Type.Range.
173 fromPositions(this._root,
174 this.start + adj, this.
175 end + adj);
176 this._writer.remove(range);
177 this._caret.setOffset(this.start
178 + adj);
179 this.undone = false;
180 return this;
181 };
182
183 /**
184  * Inserts the removed text again
185  * @param {Number[][]} shifts
186  * @returns {Type.Actions.Remove}
187  * - This instance
188 */
189 this.undo = function (shifts) {
190     var adj = this._getShiftTo(this.
191     start, shifts),
192     nodeInfo = Type.TextWalker.
193     nodeAt(this._root, this.
194     start + adj);
195     this._writer.insertHTML(nodeInfo
196     .node, nodeInfo.offset,
197     this._contents);
198     this._caret.setOffset(this.end +
199     adj);
200     this.undone = true;
201     return this;
202 };
203
204 /**
205  * Returns if a given action can
206  * be merged with this
207  * action
208  * @param {*} that
209  * @returns {boolean}
210 */
211 this.mergeable = function (that) {
212     return false; // Deactivated
213 }
214
215 range = Type.Range.
216 fromPositions(this._root,
217 this.start + adj, this.
218 end + adj);
219 this._writer.remove(range);
220 this._caret.setOffset(this.start
221 + adj);
222 this.undone = false;
223 return this;
224 };
225
226 /**
227  * Inserts the removed text again
228  * @param {Number[][]} shifts
229  * @returns {Type.Actions.Remove}
230  * - This instance
231 */
232 this.undo = function (shifts) {
233     var adj = this._getShiftTo(this.
234     start, shifts),
235     nodeInfo = Type.TextWalker.
236     nodeAt(this._root, this.
237     start + adj);
238     this._writer.insertHTML(nodeInfo
239     .node, nodeInfo.offset,
240     this._contents);
241     this._caret.setOffset(this.end +
242     adj);
243     this.undone = true;
244     return this;
245 };
246
247 /**
248  * Returns if a given action can
249  * be merged with this
250  * action
251  * @param {*} that
252  * @returns {boolean}
253 */
254 this.mergeable = function (that) {
255     return false; // Deactivated
256 }
257
258 range = Type.Range.
259 fromPositions(this._root,
260 this.start + adj, this.
261 end + adj);
262 this._writer.remove(range);
263 this._caret.setOffset(this.start
264 + adj);
265 this.undone = false;
266 return this;
267 };
268
269 /**
270  * Inserts the removed text again
271  * @param {Number[][]} shifts
272  * @returns {Type.Actions.Remove}
273  * - This instance
274 */
275 this.undo = function (shifts) {
276     var adj = this._getShiftTo(this.
277     start, shifts),
278     nodeInfo = Type.TextWalker.
279     nodeAt(this._root, this.
280     start + adj);
281     this._writer.insertHTML(nodeInfo
282     .node, nodeInfo.offset,
283     this._contents);
284     this._caret.setOffset(this.end +
285     adj);
286     this.undone = true;
287     return this;
288 };
289
290 /**
291  * Returns if a given action can
292  * be merged with this
293  * action
294  * @param {*} that
295  * @returns {boolean}
296 */
297 this.mergeable = function (that) {
298     return false; // Deactivated
299 }
300
301 range = Type.Range.
302 fromPositions(this._root,
303 this.start + adj, this.
304 end + adj);
305 this._writer.remove(range);
306 this._caret.setOffset(this.start
307 + adj);
308 this.undone = false;
309 return this;
310 };
311
312 /**
313  * Inserts the removed text again
314  * @param {Number[][]} shifts
315  * @returns {Type.Actions.Remove}
316  * - This instance
317 */
318 this.undo = function (shifts) {
319     var adj = this._getShiftTo(this.
320     start, shifts),
321     nodeInfo = Type.TextWalker.
322     nodeAt(this._root, this.
323     start + adj);
324     this._writer.insertHTML(nodeInfo
325     .node, nodeInfo.offset,
326     this._contents);
327     this._caret.setOffset(this.end +
328     adj);
329     this.undone = true;
330     return this;
331 };
332
333 /**
334  * Returns if a given action can
335  * be merged with this
336  * action
337  * @param {*} that
338  * @returns {boolean}
339 */
340 this.mergeable = function (that) {
341     return false; // Deactivated
342 }
343
344 range = Type.Range.
345 fromPositions(this._root,
346 this.start + adj, this.
347 end + adj);
348 this._writer.remove(range);
349 this._caret.setOffset(this.start
350 + adj);
351 this.undone = false;
352 return this;
353 };
354
355 /**
356  * Inserts the removed text again
357  * @param {Number[][]} shifts
358  * @returns {Type.Actions.Remove}
359  * - This instance
360 */
361 this.undo = function (shifts) {
362     var adj = this._getShiftTo(this.
363     start, shifts),
364     nodeInfo = Type.TextWalker.
365     nodeAt(this._root, this.
366     start + adj);
367     this._writer.insertHTML(nodeInfo
368     .node, nodeInfo.offset,
369     this._contents);
370     this._caret.setOffset(this.end +
371     adj);
372     this.undone = true;
373     return this;
374 };
375
376 /**
377  * Returns if a given action can
378  * be merged with this
379  * action
380  * @param {*} that
381  * @returns {boolean}
382 */
383 this.mergeable = function (that) {
384     return false; // Deactivated
385 }
386
387 range = Type.Range.
388 fromPositions(this._root,
389 this.start + adj, this.
390 end + adj);
391 this._writer.remove(range);
392 this._caret.setOffset(this.start
393 + adj);
394 this.undone = false;
395 return this;
396 };
397
398 /**
399  * Inserts the removed text again
400  * @param {Number[][]} shifts
401  * @returns {Type.Actions.Remove}
402  * - This instance
403 */
404 this.undo = function (shifts) {
405     var adj = this._getShiftTo(this.
406     start, shifts),
407     nodeInfo = Type.TextWalker.
408     nodeAt(this._root, this.
409     start + adj);
410     this._writer.insertHTML(nodeInfo
411     .node, nodeInfo.offset,
412     this._contents);
413     this._caret.setOffset(this.end +
414     adj);
415     this.undone = true;
416     return this;
417 };
418
419 /**
420  * Returns if a given action can
421  * be merged with this
422  * action
423  * @param {*} that
424  * @returns {boolean}
425 */
426 this.mergeable = function (that) {
427     return false; // Deactivated
428 }
429
430 range = Type.Range.
431 fromPositions(this._root,
432 this.start + adj, this.
433 end + adj);
434 this._writer.remove(range);
435 this._caret.setOffset(this.start
436 + adj);
437 this.undone = false;
438 return this;
439 };
440
441 /**
442  * Inserts the removed text again
443  * @param {Number[][]} shifts
444  * @returns {Type.Actions.Remove}
445  * - This instance
446 */
447 this.undo = function (shifts) {
448     var adj = this._getShiftTo(this.
449     start, shifts),
450     nodeInfo = Type.TextWalker.
451     nodeAt(this._root, this.
452     start + adj);
453     this._writer.insertHTML(nodeInfo
454     .node, nodeInfo.offset,
455     this._contents);
456     this._caret.setOffset(this.end +
457     adj);
458     this.undone = true;
459     return this;
460 };
461
462 /**
463  * Returns if a given action can
464  * be merged with this
465  * action
466  * @param {*} that
467  * @returns {boolean}
468 */
469 this.mergeable = function (that) {
470     return false; // Deactivated
471 }
472
473 range = Type.Range.
474 fromPositions(this._root,
475 this.start + adj, this.
476 end + adj);
477 this._writer.remove(range);
478 this._caret.setOffset(this.start
479 + adj);
480 this.undone = false;
481 return this;
482 };
483
484 /**
485  * Inserts the removed text again
486  * @param {Number[][]} shifts
487  * @returns {Type.Actions.Remove}
488  * - This instance
489 */
490 this.undo = function (shifts) {
491     var adj = this._getShiftTo(this.
492     start, shifts),
493     nodeInfo = Type.TextWalker.
494     nodeAt(this._root, this.
495     start + adj);
496     this._writer.insertHTML(nodeInfo
497     .node, nodeInfo.offset,
498     this._contents);
499     this._caret.setOffset(this.end +
500     adj);
501     this.undone = true;
502     return this;
503 };
504
505 /**
506  * Returns if a given action can
507  * be merged with this
508  * action
509  * @param {*} that
510  * @returns {boolean}
511 */
512 this.mergeable = function (that) {
513     return false; // Deactivated
514 }
515
516 range = Type.Range.
517 fromPositions(this._root,
518 this.start + adj, this.
519 end + adj);
520 this._writer.remove(range);
521 this._caret.setOffset(this.start
522 + adj);
523 this.undone = false;
524 return this;
525 };
526
527 /**
528  * Inserts the removed text again
529  * @param {Number[][]} shifts
530  * @returns {Type.Actions.Remove}
531  * - This instance
532 */
533 this.undo = function (shifts) {
534     var adj = this._getShiftTo(this.
535     start, shifts),
536     nodeInfo = Type.TextWalker.
537     nodeAt(this._root, this.
538     start + adj);
539     this._writer.insertHTML(nodeInfo
540     .node, nodeInfo.offset,
541     this._contents);
542     this._caret.setOffset(this.end +
543     adj);
544     this.undone = true;
545     return this;
546 };
547
548 /**
549  * Returns if a given action can
550  * be merged with this
551  * action
552  * @param {*} that
553  * @returns {boolean}
554 */
555 this.mergeable = function (that) {
556     return false; // Deactivated
557 }
558
559 range = Type.Range.
560 fromPositions(this._root,
561 this.start + adj, this.
562 end + adj);
563 this._writer.remove(range);
564 this._caret.setOffset(this.start
565 + adj);
566 this.undone = false;
567 return this;
568 };
569
570 /**
571  * Inserts the removed text again
572  * @param {Number[][]} shifts
573  * @returns {Type.Actions.Remove}
574  * - This instance
575 */
576 this.undo = function (shifts) {
577     var adj = this._getShiftTo(this.
578     start, shifts),
579     nodeInfo = Type.TextWalker.
580     nodeAt(this._root, this.
581     start + adj);
582     this._writer.insertHTML(nodeInfo
583     .node, nodeInfo.offset,
584     this._contents);
585     this._caret.setOffset(this.end +
586     adj);
587     this.undone = true;
588     return this;
589 };
590
591 /**
592  * Returns if a given action can
593  * be merged with this
594  * action
595  * @param {*} that
596  * @returns {boolean}
597 */
598 this.mergeable = function (that) {
599     return false; // Deactivated
600 }
601
602 range = Type.Range.
603 fromPositions(this._root,
604 this.start + adj, this.
605 end + adj);
606 this._writer.remove(range);
607 this._caret.setOffset(this.start
608 + adj);
609 this.undone = false;
610 return this;
611 };
612
613 /**
614  * Inserts the removed text again
615  * @param {Number[][]} shifts
616  * @returns {Type.Actions.Remove}
617  * - This instance
618 */
619 this.undo = function (shifts) {
620     var adj = this._getShiftTo(this.
621     start, shifts),
622     nodeInfo = Type.TextWalker.
623     nodeAt(this._root, this.
624     start + adj);
625     this._writer.insertHTML(nodeInfo
626     .node, nodeInfo.offset,
627     this._contents);
628     this._caret.setOffset(this.end +
629     adj);
630     this.undone = true;
631     return this;
632 };
633
634 /**
635  * Returns if a given action can
636  * be merged with this
637  * action
638  * @param {*} that
639  * @returns {boolean}
640 */
641 this.mergeable = function (that) {
642     return false; // Deactivated
643 }
644
645 range = Type.Range.
646 fromPositions(this._root,
647 this.start + adj, this.
648 end + adj);
649 this._writer.remove(range);
650 this._caret.setOffset(this.start
651 + adj);
652 this.undone = false;
653 return this;
654 };
655
656 /**
657  * Inserts the removed text again
658  * @param {Number[][]} shifts
659  * @returns {Type.Actions.Remove}
660  * - This instance
661 */
662 this.undo = function (shifts) {
663     var adj = this._getShiftTo(this.
664     start, shifts),
665     nodeInfo = Type.TextWalker.
666     nodeAt(this._root, this.
667     start + adj);
668     this._writer.insertHTML(nodeInfo
669     .node, nodeInfo.offset,
670     this._contents);
671     this._caret.setOffset(this.end +
672     adj);
673     this.undone = true;
674     return this;
675 };
676
677 /**
678  * Returns if a given action can
679  * be merged with this
680  * action
681  * @param {*} that
682  * @returns {boolean}
683 */
684 this.mergeable = function (that) {
685     return false; // Deactivated
686 }
687
688 range = Type.Range.
689 fromPositions(this._root,
690 this.start + adj, this.
691 end + adj);
692 this._writer.remove(range);
693 this._caret.setOffset(this.start
694 + adj);
695 this.undone = false;
696 return this;
697 };
698
699 /**
700  * Inserts the removed text again
701  * @param {Number[][]} shifts
702  * @returns {Type.Actions.Remove}
703  * - This instance
704 */
705 this.undo = function (shifts) {
706     var adj = this._getShiftTo(this.
707     start, shifts),
708     nodeInfo = Type.TextWalker.
709     nodeAt(this._root, this.
710     start + adj);
711     this._writer.insertHTML(nodeInfo
712     .node, nodeInfo.offset,
713     this._contents);
714     this._caret.setOffset(this.end +
715     adj);
716     this.undone = true;
717     return this;
718 };
719
720 /**
721  * Returns if a given action can
722  * be merged with this
723  * action
724  * @param {*} that
725  * @returns {boolean}
726 */
727 this.mergeable = function (that) {
728     return false; // Deactivated
729 }
730
731 range = Type.Range.
732 fromPositions(this._root,
733 this.start + adj, this.
734 end + adj);
735 this._writer.remove(range);
736 this._caret.setOffset(this.start
737 + adj);
738 this.undone = false;
739 return this;
740 };
741
742 /**
743  * Inserts the removed text again
744  * @param {Number[][]} shifts
745  * @returns {Type.Actions.Remove}
746  * - This instance
747 */
748 this.undo = function (shifts) {
749     var adj = this._getShiftTo(this.
750     start, shifts),
751     nodeInfo = Type.TextWalker.
752     nodeAt(this._root, this.
753     start + adj);
754     this._writer.insertHTML(nodeInfo
755     .node, nodeInfo.offset,
756     this._contents);
757     this._caret.setOffset(this.end +
758     adj);
759     this.undone = true;
760     return this;
761 };
762
763 /**
764  * Returns if a given action can
765  * be merged with this
766  * action
767  * @param {*} that
768  * @returns {boolean}
769 */
770 this.mergeable = function (that) {
771     return false; // Deactivated
772 }
773
774 range = Type.Range.
775 fromPositions(this._root,
776 this.start + adj, this.
777 end + adj);
778 this._writer.remove(range);
779 this._caret.setOffset(this.start
780 + adj);
781 this.undone = false;
782 return this;
783 };
784
785 /**
786  * Inserts the removed text again
787  * @param {Number[][]} shifts
788  * @returns {Type.Actions.Remove}
789  * - This instance
790 */
791 this.undo = function (shifts) {
792     var adj = this._getShiftTo(this.
793     start, shifts),
794     nodeInfo = Type.TextWalker.
795     nodeAt(this._root, this.
796     start + adj);
797     this._writer.insertHTML(nodeInfo
798     .node, nodeInfo.offset,
799     this._contents);
800     this._caret.setOffset(this.end +
801     adj);
802     this.undone = true;
803     return this;
804 };
805
806 /**
807  * Returns if a given action can
808  * be merged with this
809  * action
810  * @param {*} that
811  * @returns {boolean}
812 */
813 this.mergeable = function (that) {
814     return false; // Deactivated
815 }
816
817 range = Type.Range.
818 fromPositions(this._root,
819 this.start + adj, this.
820 end + adj);
821 this._writer.remove(range);
822 this._caret.setOffset(this.start
823 + adj);
824 this.undone = false;
825 return this;
826 };
827
828 /**
829  * Inserts the removed text again
830  * @param {Number[][]} shifts
831  * @returns {Type.Actions.Remove}
832  * - This instance
833 */
834 this.undo = function (shifts) {
835     var adj = this._getShiftTo(this.
836     start, shifts),
837     nodeInfo = Type.TextWalker.
838     nodeAt(this._root, this.
839     start + adj);
840     this._writer.insertHTML(nodeInfo
841     .node, nodeInfo.offset,
842     this._contents);
843     this._caret.setOffset(this.end +
844     adj);
845     this.undone = true;
846     return this;
847 };
848
849 /**
850  * Returns if a given action can
851  * be merged with this
852  * action
853  * @param {*} that
854  * @returns {boolean}
855 */
856 this.mergeable = function (that) {
857     return false; // Deactivated
858 }
859
860 range = Type.Range.
861 fromPositions(this._root,
862 this.start + adj, this.
863 end + adj);
864 this._writer.remove(range);
865 this._caret.setOffset(this.start
866 + adj);
867 this.undone = false;
868 return this;
869 };
870
871 /**
872  * Inserts the removed text again
873  * @param {Number[][]} shifts
874  * @returns {Type.Actions.Remove}
875  * - This instance
876 */
877 this.undo = function (shifts) {
878     var adj = this._getShiftTo(this.
879     start, shifts),
880     nodeInfo = Type.TextWalker.
881     nodeAt(this._root, this.
882     start + adj);
883     this._writer.insertHTML(nodeInfo
884     .node, nodeInfo.offset,
885     this._contents);
886     this._caret.setOffset(this.end +
887     adj);
888     this.undone = true;
889     return this;
890 };
891
892 /**
893  * Returns if a given action can
894  * be merged with this
895  * action
896  * @param {*} that
897  * @returns {boolean}
898 */
899 this.mergeable = function (that) {
900     return false; // Deactivated
901 }
902
903 range = Type.Range.
904 fromPositions(this._root,
905 this.start + adj, this.
906 end + adj);
907 this._writer.remove(range);
908 this._caret.setOffset(this.start
909 + adj);
910 this.undone = false;
911 return this;
912 };
913
914 /**
915  * Inserts the removed text again
916  * @param {Number[][]} shifts
917  * @returns {Type.Actions.Remove}
918  * - This instance
919 */
920 this.undo = function (shifts) {
921     var adj = this._getShiftTo(this.
922     start, shifts),
923     nodeInfo = Type.TextWalker.
924     nodeAt(this._root, this.
925     start + adj);
926     this._writer.insertHTML(nodeInfo
927     .node, nodeInfo.offset,
928     this._contents);
929     this._caret.setOffset(this.end +
930     adj);
931     this.undone = true;
932     return this;
933 };
934
935 /**
936  * Returns if a given action can
937  * be merged with this
938  * action
939  * @param {*} that
940  * @returns {boolean}
941 */
942 this.mergeable = function (that) {
943     return false; // Deactivated
944 }
945
946 range = Type.Range.
947 fromPositions(this._root,
948 this.start + adj, this.
949 end + adj);
950 this._writer.remove(range);
951 this._caret.setOffset(this.start
952 + adj);
953 this.undone = false;
954 return this;
955 };
956
957 /**
958  * Inserts the removed text again
959  * @param {Number[][]} shifts
960  * @returns {Type.Actions.Remove}
961  * - This instance
962 */
963 this.undo = function (shifts) {
964     var adj = this._getShiftTo(this.
965     start, shifts),
966     nodeInfo = Type.TextWalker.
967     nodeAt(this._root, this.
968     start + adj);
969     this._writer.insertHTML(nodeInfo
970     .node, nodeInfo.offset,
971     this._contents);
972     this._caret.setOffset(this.end +
973     adj);
974     this.undone = true;
975     return this;
976 };
977
978 /**
979  * Returns if a given action can
980  * be merged with this
981  * action
982  * @param {*} that
983  * @returns {boolean}
984 */
985 this.mergeable = function (that) {
986     return false; // Deactivated
987 }
988
989 range = Type.Range.
990 fromPositions(this._root,
991 this.start + adj, this.
992 end + adj);
993 this._writer.remove(range);
994 this._caret.setOffset(this.start
995 + adj);
996 this.undone = false;
997 return this;
998 };
999
1000 /**
1001  * Inserts the removed text again
1002  * @param {Number[][]} shifts
1003  * @returns {Type.Actions.Remove}
1004  * - This instance
1005 */
1006 this.undo = function (shifts) {
1007     var adj = this._getShiftTo(this.
1008     start, shifts),
1009     nodeInfo = Type.TextWalker.
1010     nodeAt(this._root, this.
1011     start + adj);
1012     this._writer.insertHTML(nodeInfo
1013     .node, nodeInfo.offset,
1014     this._contents);
1015     this._caret.setOffset(this.end +
1016     adj);
1017     this.undone = true;
1018     return this;
1019 };
1020
1021 /**
1022  * Returns if a given action can
1023  * be merged with this
1024  * action
1025  * @param {*} that
1026  * @returns {boolean}
1027 */
1028 this.mergeable = function (that) {
1029     return false; // Deactivated
1030 }
1031
1032 range = Type.Range.
1033 fromPositions(this._root,
1034 this.start + adj, this.
1035 end + adj);
1036 this._writer.remove(range);
1037 this._caret.setOffset(this.start
1038 + adj);
1039 this.undone = false;
1040 return this;
1041 };
1042
1043 /**
1044  * Inserts the removed text again
1045  * @param {Number[][]} shifts
1046  * @returns {Type.Actions.Remove}
1047  * - This instance
1048 */
1049 this.undo = function (shifts) {
1050     var adj = this._getShiftTo(this.
1051     start, shifts),
1052     nodeInfo = Type.TextWalker.
1053     nodeAt(this._root, this.
1054     start + adj);
1055     this._writer.insertHTML(nodeInfo
1056     .node, nodeInfo.offset,
1057     this._contents);
1058     this._caret.setOffset(this.end +
1059     adj);
1060     this.undone = true;
1061     return this;
1062 };
1063
1064 /**
1065  * Returns if a given action can
1066  * be merged with this
1067  * action
1068  * @param {*} that
1069  * @returns {boolean}
1070 */
1071 this.mergeable = function (that) {
1072     return false; // Deactivated
1073 }
1074
1075 range = Type.Range.
1076 fromPositions(this._root,
1077 this.start + adj, this.
1078 end + adj);
1079 this._writer.remove(range);
1080 this._caret.setOffset(this.start
1081 + adj);
1082 this.undone = false;
1083 return this;
1084 };
1085
1086 /**
1087  * Inserts the removed text again
1088  * @param {Number[][]} shifts
1089  * @returns {Type.Actions.Remove}
1090  * - This instance
1091 */
1092 this.undo = function (shifts) {
1093     var adj = this._getShiftTo(this.
1094     start, shifts),
1095     nodeInfo = Type.TextWalker.
1096     nodeAt(this._root, this.
1097     start + adj);
1098     this._writer.insertHTML(nodeInfo
1099     .node, nodeInfo.offset,
1100     this._contents);
1101     this._caret.setOffset(this.end +
1102     adj);
1103     this.undone = true;
1104     return this;
1105 };
1106
1107 /**
1108  * Returns if a given action can
1109  * be merged with this
1110  * action
1111  * @param {*} that
1112  * @returns {boolean}
1113 */
1114 this.mergeable = function (that) {
1115     return false; // Deactivated
1116 }
1117
1118 range = Type.Range.
1119 fromPositions(this._root,
1120 this.start + adj, this.
1121 end + adj);
1122 this._writer.remove(range);
1123 this._caret.setOffset(this.start
1124 + adj);
1125 this.undone = false;
1126 return this;
1127 };
1128
1129 /**
1130  * Inserts the removed text again
1131  * @param {Number[][]} shifts
1132  * @returns {Type.Actions.Remove}
1133  * - This instance
1134 */
1135 this.undo = function (shifts) {
1136     var adj = this._getShiftTo(this.
1137     start, shifts),
1138     nodeInfo = Type.TextWalker.
1139     nodeAt(this._root, this.
1140     start + adj);
1141     this._writer.insertHTML(nodeInfo
1142     .node, nodeInfo.offset,
1143     this._contents);
1144     this._caret.setOffset(this.end +
1145     adj);
1146     this.undone = true;
1147     return this;
1148 };
1149
1150 /**
1151  * Returns if a given action can
1152  * be merged with this
1153  * action
1154  * @param {*} that
1155  * @returns {boolean}
1156 */
1157 this.mergeable = function (that) {
1158     return false; // Deactivated
1159 }
1160
1161 range = Type.Range.
1162 fromPositions(this._root,
1163 this.start + adj, this.
1164 end + adj);
1165 this._writer.remove(range);
1166 this._caret.setOffset(this.start
1167 + adj);
1168 this.undone = false;
1169 return this;
1170 };
1171
1172 /**
1173  * Inserts the removed text again
1174  * @param {Number[][]} shifts
1175  * @returns {Type.Actions.Remove}
1176  * - This instance
1177 */
1178 this.undo = function (shifts) {
1179     var adj = this._getShiftTo(this.
1180     start, shifts),
1181     nodeInfo = Type.TextWalker.
1182     nodeAt(this._root, this.
1183     start + adj);
1184     this._writer.insertHTML(nodeInfo
1185     .node, nodeInfo.offset,
1186     this._contents);
1187     this._caret.setOffset(this.end +
1188     adj);
1189     this.undone = true;
1190     return this;
1191 };
1192
1193 /**
1194  * Returns if a given action can
1195  * be merged with this
1196  * action
1197  * @param {*} that
1198  * @returns {boolean}
1199 */
1200 this.mergeable = function (that) {
1201     return false; // Deactivated
1202 }
1203
1204 range = Type.Range.
1205 fromPositions(this._root,
1206 this.start + adj, this.
1207 end + adj);
1208 this._writer.remove(range);
1209 this._caret.setOffset(this.start
1210 + adj);
1211 this.undone = false;
1212 return this;
1213 };
1214
1215 /**
1216  * Inserts the removed text again
1217  * @param {Number[][]} shifts
1218  * @returns {Type.Actions.Remove}
1219  * - This instance
1220 */
1221 this.undo = function (shifts) {
1222     var adj = this._getShiftTo(this.
1223     start, shifts),
1224     nodeInfo = Type.TextWalker.
1225     nodeAt(this._root, this.
1226     start + adj);
1227     this._writer.insertHTML(nodeInfo
1228     .node, nodeInfo.offset,
1229     this._contents);
1230     this._caret.setOffset(this.end +
1231     adj);
1232     this.undone = true;
1233     return this;
1234 };
1235
1236 /**
1237  * Returns if a given action can
1238  * be merged with this
1239  * action
1240  * @param {*} that
1241  * @returns {boolean}
1242 */
1243 this.mergeable = function (that) {
1244     return false; // Deactivated
1245 }
1246
1247 range = Type.Range.
1248 fromPositions(this._root,
1249 this.start + adj, this.
1250 end + adj);
1251 this._writer.remove(range);
1252 this._caret.setOffset(this.start
1253 + adj);
1254 this.undone = false;
1255 return this;
1256 };
1257
1258 /**
1259  * Inserts the removed text again
1260  * @param {Number[][]} shifts
1261  * @returns {Type.Actions.Remove}
1262  * - This instance
1263 */
1264 this.undo = function (shifts) {
1265     var adj = this._getShiftTo(this.
1266     start, shifts),
1267     nodeInfo = Type.TextWalker.
1268     nodeAt(this._root, this.
1269     start + adj);
1270     this._writer.insertHTML(nodeInfo
1271     .node, nodeInfo.offset,
1272     this._contents);
1273     this._caret.setOffset(this.end +
1274     adj);
1275     this.undone = true;
1276     return this;
1277 };
1278
1279 /**
1280  * Returns if a given action can
1281  * be merged with this
1282  * action
1283  * @param {*} that
1284  * @returns {boolean}
1285 */
1286 this.mergeable = function (that) {
1287     return false; // Deactivated
1288 }
1289
1290 range = Type.Range.
1291 fromPositions(this._root,
1292 this.start + adj, this.
1293 end + adj);
1294 this._writer.remove(range);
1295 this._caret.setOffset(this.start
1296 + adj);
1297 this.undone = false;
1298 return this;
1299 };
1300
1301 /**
1302  * Inserts the removed text again
1303  * @param {Number[][]} shifts
1304  * @returns {Type.Actions.Remove}
1305  * - This instance
1306 */
1307 this.undo = function (shifts) {
1308     var adj = this._getShiftTo(this.
1309     start, shifts),
1310     nodeInfo = Type.TextWalker.
1311     nodeAt(this._root, this.
1312     start + adj);
1313     this._writer.insertHTML(nodeInfo
1314     .node, nodeInfo.offset,
1315     this._contents);
1316     this._caret.setOffset(this.end +
1317     adj);
1318     this.undone = true;
1319     return this;
1320 };
1321
1322 /**
1323  * Returns if a given action can
1324  * be merged with this
1325  * action
1326  * @param {*} that
1327  * @returns {boolean}
1328 */
1329 this.mergeable = function (that) {
1330     return false; // Deactivated
1331 }
1332
1333 range = Type.Range.
1334 fromPositions(this._root,
1335 this.start + adj, this.
1336 end + adj);
1337 this._writer.remove(range);
1338 this._caret.setOffset(this.start
1339 + adj);
1340 this.undone = false;
1341 return this;
1342 };
1343
1344 /**
1345  * Inserts
```

```

1  'use strict';
2
3  var Type = require('./core');
4
5  /**
6   * Creates a new Type action
7   * @param {*} sourceId - An
8   *   arbitrary key identifying the
9   *   of this action
10  * @param {Type} type - A type
11  *   instance on which the action
12  *   should be executed
13  * @param {Number} start - The
14  *   character offset from which
15  *   the
16  *   text should be formatted
17  * @param {Number} end - The
18  *   character offset to which the
19  *   text should be formatted
20  * @param {Element[]} nodes - The
21  *   initial elements that have
22  *   been
23  *   affected by performing this
24  *   action
25  * @param {Number} tag - The tag the
26  *   text should be formatted
27  * with
28  * @param {boolean} [undone] - The
29  *   state of this action
30  * @constructor
31  */
32 Type.Actions.Format = function (
33   sourceId, type, start, end, tag
34   , nodes, undone) {
35   this.sourceId = sourceId;
36   this.undone = undone || false;
37   this._formatter = type.
38     getFormatter();
39   this._caret = type.getCaret();
40   this._root = type.getRoot();
41   this._start = start;
42   this._end = end;
43   this._tag = tag;
44 }
45
46 Type.Actions.Format.fromRange =
47   function (sourceId, type, range
48     , tag, nodes) {
49   var bookmark = range.save(type.
50     getRoot());
51   return new Type.Actions.Format(
52     sourceId, type, bookmark.
53     start, bookmark.end, tag,
54     nodes);
55 };
56
57 module.exports = Type.Actions.Format;
58
59 Listing 31: actions/format.js

```

```

1  'use strict';
2
3  var Type = require('./core');
4
5  (function () {
6
7    /**
8     * Returns the offset of the caret
9     * type.caret()
10
11    * Show the caret
12    * type.caret('show')
13
14    * Hides the caret
15    * type.caret('hide')
16
17    * Moves the caret to the 10th
18    * character
19    * type.caret(10)
20
21    * Convenience function for type.
22    * select(10, 20)
23    * type.caret(10, 20)
24
25    * @param {...*} params - Various
26    * parameters are possible.
27    * See examples in the block
28    * comment.
29    * @returns {Type} - The editor
30    * instance
31
32    * this.caret = function (params) {
33
34    // type.caret() todo was ist bei
35    // selection?
36    if (arguments.length) {
37      return this._caret.getOffset()
38    }
39
40    // type.caret('show')
41    if (arguments[0] === 'show') {
42      this._caret.show();
43      return this;
44    }
45
46    // type.caret('hide')
47    if (arguments[0] === 'hide') {
48      this._caret.hide();
49      return this;
50    }
51
52    // type.caret(10)
53    if (arguments.length === 1 &&
54      typeof arguments[0] === "number") {
55      this._caret.setOffset(
56        arguments[0]);
57      return this;
58    }
59
60    // type.caret(10, 20)
61    if (arguments.length === 2) {
62      return this.selection(
63        arguments[0], arguments
64        [1]);
65    }
66
67    return this;
68  };
69
70  /**
71   * Same as type.selection('text')
72   * type.selection()
73
74   * Returns the unformatted (plain)
75   * contents of the current
76   * selection
77   * type.selection('text')
78
79   * Return the currently selected
80   * HTML
81   * type.selection('html')
82
83   * Convenience function for type.
84   * caret(10)
85   * type.selection(10)
86
87   *
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

* Selects characters 10 to 20
* type.selection(10, 20)
*
* Select an element
* type.selection(element)
*
* Creates a selection between 2
* elements
* type.selection(element1,
  element2)
*
* Creates a selection between the
  first and last element in
  the jQuery Collection
* type.selection(jQueryCollection
  )
*
* Returns an object that can be
  passed to type.selection('
  restore') to store and
  recreate a selection
* type.selection('save')
*
* Takes an object returned by
  type.selection('save') as a
  second argument to recreate
  a stored selection
* type.selection('restore', sel)
*
* @param {...*} params - Various
  parameters are possible.
  See examples in the block
  comment.
* @returns {Type} - The editor
  instance
* this.selection = function (params)
  {
  // type.selection() || type.
  selection('text')
  if (!arguments.length ||
    arguments[0] === 'text') {
    return Type.Range.
      fromCurrentSelection().
      text();
  }

```

```

101     }
102
103     // type.selection('html')
104     if (arguments[0] === 'html') {
105         return Type.Range.
106             fromCurrentSelection().
107             html();
108     }
109     // type.selection(10)
110     if (arguments.length === 1 &&
111         typeof arguments[0] === "
112         number") {
113         return this.caret(arguments
114             [0]);
115     }
116     // type.selection(10, 20)
117     if (arguments.length === 2 &&
118         typeof arguments[0] === "
119         number") {
120         new Type.Range(this.root,
121             arguments[0], arguments
122             [1]).select();
123         return this;
124     }
125     // type.selection(element1,
126         element2)
127     if (arguments.length === 2 &&
128         DomUtil.isNode(arguments))
129     {
130         new Type.Range(arguments[0],
131             arguments[1]).select();
132         return this;
133     }
134 }
135
136 // type.selection() || type.
137 // selection([Array])
138 if (arguments[0].jQuery) {
139     new Type.Range(arguments[0],
140         arguments[1]).select();
141     return this;
142 }
143 // type.selection('save')
144 if (arguments[0] === 'save') {
145     return Type.Range.
146         fromCurrentSelection().
147         save();
148 }
149 // type.selection('restore', sel)
150 if (arguments[0] === 'restore') {
151     return Type.Range.
152         fromCurrentSelection().
153         restore(arguments[1]);
154     return this;
155 }
156 /**
157 * Inserts plain text at the caret
158 * 's position, regardless if
159 * str contains html. Will
160 * overwrite the current
161 * selection if there is one.
162 */
163 * type.insert(str)* selection if
164 * there is one.
165 *
166 * Inserts formatted text at the
167 * caret 's position. Will
168 * overwrite the current
169 * selection if there is one.
170 *
171 * type.insert('html', str, 10)
172 * but inserts formatted text
173 * given as html string
174 * type.insert('html', str, 10).
175 *
176 * @param {...*} params - Various
177 * parameters are possible.
178 * See examples in the block
179 * comment.
180 * @returns {Type} - The editor
181 * instance
182 */
183 this.insert = function (params) {
184     // type.insert(str)
185     if (arguments.length === 1) {
186         this.getInput().getContent().
187             insert(this.getCaret().
188                 getOffset(), arguments
189                 [0]);
190         return this;
191     }
192     // type.insert(str, 'text')
193     if (arguments.length === 2 &&
194         arguments[1] === 'text') {
195         // this._writer.insertText(
196             arguments[0]);
197         this.getInput().getContent().
198             insert(this.getCaret().
199                 getOffset(), arguments
200                 [0]);
201         return this;
202     }
203     // type.insert(str, 10)
204     if (arguments.length === 2 &&
205         typeof arguments[1] ===
206         'number') {
207         this._writer.insertText(
208             arguments[0], arguments
209             [1]);
210     }
211 }

```

```

187         [1]);
188         return this;
189     }
190     // type.insert('html', str, 10)
191     if (arguments.length === 3 &&
192         arguments[0] === 'html') {
193         this._writer.insertHTML(
194             arguments[1], arguments
195             [2]);
196         return this;
197     }
198     return this;
199 };
200 /**
201  * Formats the currently selected
202  * text with the given tag.
203  * type.format(tagName, [...params
204  *   ]) * E.g. use type.cmd('
205  *   strong') to format the
206  *   currently selected text bold
207  *
208  * Applies type.format to a
209  * specific text range
210  * type.format(startOffset,
211  *   endOffset, tagName, [...
212  *   params])
213  *
214  * @param {...*} params - Various
215  *   parameters are possible.
216  *   See examples in the block
217  *   comment.
218  * @returns {Type} - The editor
219  *   instance
220  */
221     this.format = function (params) {
222         var sel, range;
223         if (arguments.length === 1) {
224             sel = this._selection.save();
225             this.getInput().getContent().
226                 save();
227         }
228         if (arguments.length < 2) {
229             sel = this._selection;
230             range = Type.Range.
231                 fromPositions(this.
232                     getRoot(), arguments[1],
233                     arguments[2]);
234             this.getInput().getContent().
235                 format(arguments[0],
236                     range);
237             return this;
238         }
239         return this;
240     };
241     /**
242  * Deletes the currently selected
243  * text. Does nothing if there
244  * is no selection.
245  * type.remove()
246  *
247  * Removes a number of characters
248  * from the caret's position. A
249  * negative number will remove
250  * characters left
251  * of the caret, a positive number
252  * from the right. If there is
253  * a selection, the characters
254  * will be removed
255  * from the end of the selection.
256  * type.remove(numChars)
257  *
258  * Will remove characters between
259  * the given offsets
260  * type.remove(startOffset,
261  *   endOffset)
262  *
263  * @param {...*} params - Various
264  *   parameters are possible.
265  *   See examples in the block
266  *   comment.
267  * @returns {Type} - The editor
268  *   instance
269  */
270     this.remove = function (params) {
271         var sourceId = this.getInput().
272             getSourceId();
273         this.undoManager().undo(
274             sourceId, steps);
275         return this;
276     };
277     /**
278  * Reapplies a revoked input
279  * @param {Number} [steps] - The
280  *   number of actions to reapply
281  * @returns {Type} - The editor
282  *   instance
283  */
284     this.reapply = function (steps) {
285         var sourceId = this.getInput().
286             getSourceId();
287         this.undoManager().undo(
288             sourceId, steps);
289         return this;
290     };
291     /**
292  * Reapplies a revoked input
293  * @param {Number} [steps] - The
294  *   number of actions to reapply
295  * @returns {Type} - The editor
296  *   instance
297  */
298     this.reapply = function (steps) {
299         var sourceId = this.getInput().
300             getSourceId();
301         this.undoManager().undo(
302             sourceId, steps);
303         return this;
304     };

```

```

277 this.redo = function (steps) {
278   var sourceId = this.getInput().
280   getContent().getSourceId();
281   this.getUndoManager().redo(
282
283     sourceId, steps);
284   return this;
285 };

```

Listing 32: core_api.js

```

31  * with an
32  * Etherpad server.
33  *
34  * @type {{host: string, port:
35  *       number, rootPath: string,
36  *       apikey: null}}
37  */
38 this._defaultOptions = {
39   host      : 'localhost',
40   port      : 9001,
41   rootPath  : '/api/1.2.1/',
42 };
43
44 /**
45  * Sets the options to be used for
46  * communicating with an
47  * Etherpad server. Takes either a
48  * plain object or a key
49  * value combination to set a
50  * single, specific option.
51  * In the latter case, the key
52  * must be a {string}.
53  *
54  * @param {(string|Object)}
55  *       options - Either a plain
56  *       object
57  *       with keys and values to be
58  *       set or a string that will
59  *       be used as a name for a
60  *       option. If you pass a string
61  *       , pass a second parameter to
62  *       set that option or no
63  *       second parameter to
64  *       retrieve that option.
65  * @param {*} [value] - If the
66  *       first parameter is a string,
67  *       this value will be set to
68  *       the key of the given first
69
70  * parameter. Any arbitrary
71  * value can be set.
72  * @returns {Type*} Returns the
73  * type instance if you set an
74  * option or the according
75  * value if you get an option
76  */
77 this.options = function (options,
78   value) {
79   // Load default options if there
80   // are no instance options
81   // yet
82   this._options = this._options ||
83     Type.Utilities.extend({},
84       this._defaultOptions);
85
86   // Pass a single option name to
87   // fetch it
88   if (typeof options === "string"
89     && arguments.length === 1)
90   {
91     return this._options[options];
92   }
93
94   // Pass an option name and a
95   // value to set it
96   if (typeof options === "string"
97     && arguments.length === 2)
98   {
99     options = {options: value};
100  }
101
102  // Pass an object of key-values
103  // to set them
104  if (typeof options === "object")
105  {
106    Type.Utilities.extend(this,
107      options);
108  }

```

```

76         _options, options);
77     }
78     // Chaining / Returning data
79     return arguments.length ? this :
80         this._options;
81 }
82
83 /**
84  * Getter for the Type instance
85  * @returns {Type}
86  */
87 this.getType = function () {
88     return this._type;
89 };
90
91 /**
92  * Getter for the Etherpad client
93  * @returns {Type.Etherpad.Client}
94  */
95 this.getClient = function () {
96     return this._client;
97 };
98
99 /**
100  * Getter for the Etherpad content
101  * @returns {Type.Etherpad.Content}
102     }
103 this.getContent = function () {
104     return this._content;
105 };
106
107 /**
108  * Will load the pad contents from
109     an Etherpad connection
110     message
111     to the Type editor contents.
112  * @param {{attrs: string, text:
113     string}} contents - The
114     contents
115     of the editor sent by the
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```



```

167     appendix = '$';
168     return prefix + body +
169         appendix;
170 }
171 }).call (Type.Etherpad.prototype);
172 /**
173  * Creates a new Type instance
174  * connected to an Etherpad
175  *
176  * @param {{}}|Element} options - The
177  * options you would pass to
178  * instantiate a
179  * Type instance
180  * @param {{}} options.etherpad -
181  * The options for the Type.
182  * Etherpad
183  * constructor
184  * @param {{}}|string} [etherpadOpts]
185  * - Either the parameters for
186  * the
187  * Type.Etherpad constructor or
188  * a pad name as a string
189  * @param {string} [server] - The
190  * URL for the Etherpad server
191  * @constructor
192 */
193 Type.fromEtherpad = function (options
194 , etherpadOpts, server) {
195     options = Type.Etherpad.
196
197     use strict';
198     var Type = require ( './.../core ');
199
200     /**
201      * Creates a new Type.Etherpad.Util
202      * instance
203      * Contains utility methods for Type
204      * .Etherpad
205      * @constructor
206      */
207
208     prepareOptions (options,
209         etherpadOpts, server);
210     var type = new Type (options);
211     new Type.Etherpad (type);
212     return type;
213 };
214 /**
215  * Used for the Type.fromEtherpad
216  * constructor to process its
217  * parameters
218  * @param {{}} options - The options
219  * a
220  * you would pass to instantiate
221  * a
222  * Type instance
223  * @param {{}} options.etherpad -
224  * The options for the Type.
225  * Etherpad
226  * constructor
227  * @param {{}}|string} [etherpadOpts]
228  * - Either the parameters for
229  * the
230  * Type.Etherpad constructor or
231  * a pad name as a string
232  * @param {string} [server] - The
233  * URL for the Etherpad server
234  * @returns {{}}
235 */
236 Type.Etherpad.prepareOptions =
237     function (options, etherpadOpts
238 , server) {
239
240     Type.Etherpad.Util = function () {
241     };
242     (function () {
243         /**
244          * Replaces newlines with <br />
245          * tags
246          * @param {string} str - The
247          * original string containing
248
249         options = options || {};
250         etherpadOpts = etherpadOpts || {};
251         if (Type.DomUtilities.isNode(
252             options)) {
253             options = { el: options };
254         }
255         if (arguments.length === 3) {
256             etherpadOpts = { pad:
257                 etherpadOpts, server:server
258             };
259         }
260         if (typeof etherpadOpts === '
261             string') {
262             etherpadOpts = { pad:
263                 etherpadOpts };
264         }
265         options.etherpad = etherpadOpts;
266         return options;
267     };
268     module.exports = Type.Etherpad;
269
270 Listing 33:
271 /plugins/etherpad/type.etherpad.js
272
273     newlines
274     * @returns {string} - The altered
275     string containing <br />
276     tags
277     */
278     Type.Etherpad.Util.nl2br =
279         function (str)
280     {
281         return (str + '').replace (/(\^>
282             |r\n|?)|(r\n|n\r|n\r|n)/g,
283             '$1<br ' + '>$2');
284     };

```

```

24  };
25
26  /**
27   * Returns a random string
   starting with 't,' that can
   be used as a token for
   server.
28   * connecting to an Etherpad
29   *
30   * @returns {string}
31   */
32   Type.Etherpad.Util.getRandomToken
   = function ()
33
34   {
35     var chars = '0123456789
36     ABCDEFGHIJKLMNOPQRSTUVWXYZ
37     ',
38     stringLength = 20,
39     randomNumber,
40     str = '';
41     for (var i = 0; i < stringLength
42         ; i++)
43     {
44       randomNumber = Math.floor(Math.
45       random() * chars.length);
46       str += chars.substring(
47       randomNumber, randomNumber +
48       1);
49       return 't.' + str;
50     }
51   }.call(Type.Etherpad.Util.prototype
52   );
53   module.exports = Type.Etherpad.Util;
54   Listing 34: /plugins/etherpad/util.js

```

```

1  'use strict';
2
3  var Type = require('.../.../core');
4
5  /**
6   * Creates a new Type.Etherpad.
   Client instance
7   *
8   * @param etherpad
9   * @constructor
10  */
11  Type.Etherpad.Client = function (
   etherpad) {
12    this._etherpad = etherpad;
13    this._msgHandlers = {};
14    this._accepted = true;
15    this._lastSent = Date.now();
16    this._changeset = new Type.
   Etherpad.Changeset();
17    this.registerMessageHandler('
   ACCEPT_COMMIT', this.
   _acceptCommit.bind(this));
18  };
19
20  (function () {
21
22    /**
23     * The default URL the client
   connects to if no URL has
   been set
24     * @type {string}
25     * @private
26     */
27     this._defaultUrl = 'http://
   localhost:9001/';
28
29     /**
30     * The interval in which changesets
   will be sent
31     * @type {number}
32     * @private
33     */
34     this._debounceTime = 0;
35
36     /**
37     * Connects to an Etherpad server
38     * @returns {Type.Etherpad.Client}
39     * _ This instance
40     */
41     this._connect = function () {
42       this._socket = io.connect(this.
   _url(), this.
   _socketIoOptions());
43       this._socket.once('connect',
   this._sendClientReady.bind(

```

```

   this));
44   this._socket.on('message', this.
   _handleMessage.bind(this));
45   return this;
46 };
47
48 /**
49  * Sets a function that will be
   called when this client
   connects to
   * a server. The pad contents from
   the server will be passed
   to the
   * handler.
50  * @param {Function} handler - The
   function that will be
   called
51  * @returns {Type.Etherpad.Client}
   _ This instance
52  */
53   this._onInit = function (handler) {
54     this._onInitHandler = handler;
55     return this;
56   };
57
58 /**
59  * Registers a handler that will
   be called for a given
   message
60  */

```

```

62  * @param {string} msg - The
63  message on which the handler
64  should be called
65  * @param {Function} handler - The
66  handler that should be
67  called
68  * @returns {Type.Etherpad.Client}
69  - This instance
70
71  this.registerMessageHandler =
72  function (msg, handler) {
73  this._msgHandlers[msg] = this._
74  msgHandlers[msg] || [];
75  this._msgHandlers[msg].push(
76  handler);
77  return this;
78
79  /**
80  * Removes a handler for a given
81  message
82  * @param {string} msg - The
83  message on which the handler
84  is called
85  * @param {Function} handler - The
86  handler that should be
87  removed
88  * @returns {Type.Etherpad.Client}
89  - This instance
90
91  this.unregisterMessageHandler =
92  function (msg, handler) {
93  var index;
94  if (this._msgHandlers[msg]) {
95  index = this._msgHandlers[msg]
96  .indexOf(handler);
97  if (index > -1) {
98  this._msgHandlers[msg].
99  splice(index, 1);
100  }
101  return this;
102
103  /**
104  * @param {string} msg - The
105  message on which the handler
106  is called
107  * @param {Function} handler - The
108  handler that should be
109  removed
110  * @returns {Type.Etherpad.Client}
111  - This instance
112
113  this.unregisterMessageHandler =
114  function (msg, handler) {
115  var index;
116  if (this._msgHandlers[msg]) {
117  index = this._msgHandlers[msg]
118  .indexOf(handler);
119  if (index > -1) {
120  this._msgHandlers[msg].
121  splice(index, 1);
122  }
123  return this;
124
125  /**
126  * @param change
127  this.send = function (change) {
128  var root, changestr;
129  this._changeset._mergeOrPush(
130  change);
131  if (this._accepted && Date.now()
132  - this._debounceTime >
133  this._lastSent) {
134  this._accepted = false;
135  root = this._etherpad.
136  getContent().getRoot();
137  changestr = this._changeset.
138  getString(root);
139  this._sendChangeset(changestr)
140  ;
141  this._changeset = new Type.
142  Etherpad.Changeset();
143  this._lastSent = Date.now();
144  }
145  return this;
146
147  /**
148  * @param msg
149  * @private
150  this._acceptCommit = function (msg)
151  {
152  this._etherpad.getContent().
153  setRevision(msg.newRev);
154  this._accepted = true;
155  };
156
157  /**
158  * @param changeset

```

```

159 // Required variables
160 var msg = response.data.type,
161     len, i;
162 // Dev code
163 Type.Development.debug('message
164     Type, response);
165 // This message will be received
166     when connecting to the
167     server
168 if (response.type === '
169     CLIENT_VARS') {
170     this._init(response.data);
171     return this;
172 }
173 // Notify developers on
174     unhandled messages from the
175     server
176 if (!this._msgHandlers[msg]) {
177     Type.Development.debug('
178         Unhandled etherpad
179         message', response);
180     return this;
181 }
182 // For all other messages call
183     the according message
184     handlers
185 len = this._msgHandlers[msg].
186     length;
187 for (i = 0; i < len; i += 1) {
188     this._msgHandlers[msg][i](
189         response.data);
190 }
191 // Chaining
192 return this;
193 };
194 /**
195  * Will be called when this client
196  * successfully connected to
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Client ;
 Listing /plugins/etherpad/client.js 35:

```

1 'use strict';
2
3 var Type = require('.../core');
4
5 /**
6  * Creates a new Type.Etherpad.
7  * Content instance
8  * @param {Type.Etherpad} etherpad -
9  *   The Etherpad instance this
10  *   class will manage content
11  *   collaboration for
12  * @constructor
13  */
14 Type.Etherpad.Content = function (
15   etherpad) {
16   this._client = etherpad.getClient
17   ();
18   this._localCaret = etherpad.
19   getType().getCaret();
20   this._typeContent = new Type.
21   Content(etherpad.getType());
22   this._root = this._typeContent.
23   getRoot();
24   this._client.
25   registerMessageHandler('
26   NEW_CHANGES', this.
27   updateContent.bind(this));
28 };
29
30 (function () {
31
32   /**
33    * Applies a change to the editor',
34    s contents
35    * @param {{newRev: number,
36
37   changeset: string, apool
38   :{}} data - The data
39   received from a NEW_CHANGES
40   message
41   {Type.Etherpad.Content
42   } - This instance
43   */
44   this.updateContent = function(data
45   ) {
46     this.revision = data.newRev;
47     this.applyChangeset(data.
48     changeset, data.apool);
49     return this;
50   };
51   /**
52    * Applies a serialized changeset
53    to the editor's contents
54    * @param {string} changesetString
55    - A serialized Changeset
56    * @returns {*}
57    */
58   this.applyChangeset = function (
59     changesetString, apool) {
60     var changeset = new Type.
61     Etherpad.Changeset.
62     fromString(changesetString,
63     apool, this._root.
64     textContent);
65     changeset.apply(this.
66     _typeContent, this.
67     _localCaret);
68     return this;
69   };
70   /**
71    * Getter for the document
72
73   revision
74   * @returns {number}
75   */
76   this.getRevision = function () {
77     return this.revision;
78   };
79   /**
80    * Setter for the document
81    revision
82    * @returns {Type.Etherpad.Content
83    } - This instance
84    */
85   this.setRevision = function (rev)
86   {
87     this.revision = rev;
88     return this;
89   };
90   /**
91    * Getter for the root element
92    * @returns {number}
93    */
94   this.getRoot = function () {
95     return this._root;
96   };
97   }.call(Type.Etherpad.Content.
98   prototype);
99
100 module.exports = Type.Etherpad.
101 Content;
102
103 Listing /plugins/etherpad/content.js 36:

```

```

1  'use strict';
2
3  var Type = require('.../.../core');
4
5  /**
6   * Creates a new Type.Etherpad.
7   *   Changeset instance
8   *
9   * @constructor
10  Type.Etherpad.Changeset = function
11  () {
12    this._stack = [];
13  };
14  (function () {
15    /**
16     * Matches a changeset to an array
17     * of results for
18     * an each operation
19     *
20     * @type {RegExp}
21     * @private
22     */
23    this._changesetRegex = /(?(?:\[0-9
24      a-z|+)*)(?:\[0-9a-z|+))
25      ?([+-=])([0-9a-z|+)]\?|/g;
26
27    /**
28     * Returns a serialized changeset
29     * string based on the length
30     * of
31     * a given string or the text
32     * contents of a given element
33
34     * @param {string|Element} base -
35     *   Either a string or an
36     *   element
37     * @returns {string}
38     */
39    this.getString = function (base) {
40      var serializer = new Type.
41        Etherpad.
42        ChangesetSerializer(this);
43
44      return serializer.getString(base
45        );
46    };
47
48    /**
49     * Applies this changeset to a
50     * given content
51
52     * @param {Type.Content} content -
53     *   The content this changeset
54     *   should be applied to
55     * @param {Type.Caret} localCaret
56     *   - The local user's caret
57     * @returns {Type.Etherpad.
58     *   Changeset} - This instance
59
60     */
61    this.apply = function (content,
62      localCaret) {
63      var i, len = this._stack.length;
64      for (i = 0; i < len; i += 1) {
65        this._stack[i].apply(content,
66          localCaret);
67      }
68      return this;
69    };
70
71    /**
72     * Returns the indices of newlines
73     * in a string
74
75     * @param {string} str - The
76     *   string to return the indexes
77     *   of newlines for
78     * @returns {number[]} - An array
79     *   in indexes of the newlines
80     *   in the text
81     * @private
82     */
83    this._getNlIndices = function (str
84      ) {
85      var regex = /\n|/gi, result,
86        indices = [];
87      while ((result = regex.exec(str
88        ))) {
89        indices.push(result.index);
90      }
91    }
92  }

```

```

93     return indices;
94 };
95
96 /**
97  * Getter for the operation stack
98  * @returns {Array}
99  */
100 this.getStack = function () {
101     return this._stack;
102 };
103
104 /**
105  * @param {{ absolute: number,
106  * stack: number[] }} offset -
107  * An object
108  * containing offset
109  * information
110  * @param {string} charbank - The
111  * charbank of a string
112  * changeset
113  * @param {{attrs: string,
114  * operator: string, value:
115  * string, nl: number}} match
116  * A match as returned by
117  * _parseMatch
118  * @param {Object} apool - An
119  * Etherpad attribute pool
120  * @param {number[][]} nlIndices -
121  * @private
122  */
123 this._addMatchToStack = function (
124     offset, charbank, match,
125     apool, nlIndices) {
126
127     var delta;
128
129     this._mergeOrPush(this._
130         _createFromMatch(offset,
131             charbank, match, apool));
132
133     if (match.operator === '=' ) {
134         delta = parseInt(match.value,
135             36);
136         delta += match.nl ? 1 : 0;
137
138         return indices;
139     }
140
141     /**
142      * Getter for the operation stack
143      * @returns {Array}
144      */
145     this.getStack = function () {
146         return this._stack;
147     };
148
149     /**
150      * @param {{ absolute: number,
151      * stack: number[] }} offset -
152      * An object
153      * containing offset
154      * information
155      * @param {string} charbank - The
156      * charbank of a string
157      * changeset
158      * @param {{attrs: string,
159      * operator: string, value:
160      * string, nl: number}} match
161      * A match as returned by
162      * _parseMatch
163      * @param {Object} apool - An
164      * Etherpad attribute pool
165      * @param {number[][]} nlIndices -
166      * @private
167      */
168     this._createFromMatch = function (
169         offset, charbank, match,
170         apool) {
171
172         var attrs = this._
173             _getAttributesFromMatch(
174                 match, apool);
175
176         switch (match.operator) {
177             case '=':
178                 return Type.Etherpad.
179                     Changeset.Changes.
180                     Command.fromAPool(apool
181                         );
182             case '-':
183                 return this._
184                     _operationOrMovement(
185                         offset, charbank, match
186                         , attrs);
187             case '+':
188                 return new Type.Etherpad.
189                     Changeset.Changes.
190                     Insertion(offset,
191                         absolute, charbank);
192             case '-':
193                 return Type.Etherpad.
194                     Changeset.Changes.
195                     Removal.fromMatch(
196
197         offset.absolute += delta;
198         offset.stack.push(offset);
199     }
200
201     return this;
202 };
203
204 /**
205  * @param offset
206  * @param charbank
207  * @param match
208  * @param attrs
209  * @returns {Type.Etherpad.
210  * Changeset.Changes.Formatting}
211  */
212 @private
213 this._operationOrMovement =
214     function (offset, charbank,
215         match, attrs) {
216         if (!attrs.length) {
217             return Type.Etherpad.Changeset.
218                 Changes.Movement.
219                 fromOffsetObject(offset,
220                     match);
221         } else {
222             return Type.Etherpad.Changeset.
223                 Changes.Formatting.
224                 fromAttrs(attrs, offset.
225                     absolute, match);
226         }
227     };
228
229 /**
230  * @param {Type.Etherpad.Changeset.
231  * Changes.Change} change - A
232  * change
233  * instance or an inheriting
234  * class
235  * @returns {Type.Etherpad.
236  * Changeset.Changes}
237  */
238 @returns {Type.Etherpad.
239     Changeset.Changes}

```

```

182     Changeset} - This instance
183     */
184     this._mergeOrPush = function (
185         change) {
186         var last = this._stack[this._
187             _stack.length - 1];
188         if (!!last && last.mergable(
189             change)) {
190             last.merge(change);
191         } else if (!(change instanceof
192             Type.Etherpad.Changeset.
193             Changes.Movement)) {
194             this._stack.push(change);
195         }
196         return this;
197     };
198     /**
199     * Parses a regex match and
200     * returns a readable object
201     */
202     @param {Array|{index: number,
203         input: string}} match - A
204         RegEx match
205     * @returns {{attrs: string,
206         operator: string, value:
207         string}}
208     */
209     this._parseMatch = function (match)
210     {
211         if (match.index === this._
212             changesetRegex.lastIndex
213             ++);
214         this._changesetRegex.lastIndex
215         ++;
216         if (match[0] === '')
217             return false;
218     }
219 }
220
221 // Returns the attributes from a
222 // match and an apool
223
224 * @param {{attrs: string,
225     operator: string, value:
226     string}} match - A
227     _parseMatch
228 * @param {{numToAttrib: array}}
229     apool - An attribute pool
230     from an
231     Etherpad server
232 * @returns {*}
233 * @private
234 */
235 this._getAttributesFromMatch =
236     function (match, apool) {
237         var i;
238         if (!match.attrs.length) return
239         [];
240         i = parseInt(match.attrs.substr
241             (1));
242         return [apool.numToAttrib[i]]
243     };
244
245 // Reads the charbank from a
246 // changeset string
247
248 * @param {string} str - A
249     serialized changeset
250 * @returns {string|null} - The
251     charbank or null
252 * if there is no charbank
253 * @private
254 }
255
256 // Returns the attributes from a
257 // match and an apool
258
259 * @param {{attrs: string,
260     operator: string, value:
261     string}} match - A
262     _parseMatch
263 * @param {{numToAttrib: array}}
264     apool - An attribute pool
265     from an
266     Etherpad server
267 * @returns {*}
268 * @private
269 */
270 this._getAttributesFromMatch =
271     function (match, apool) {
272         var i;
273         if (!match.attrs.length) return
274         [];
275         i = parseInt(match.attrs.substr
276             (1));
277         return [apool.numToAttrib[i]]
278     };
279
280 // Reads the charbank from a
281 // changeset string
282
283 * @param {string} str - A
284     serialized changeset
285 * @returns {string|null} - The
286     charbank or null
287 * if there is no charbank
288 * @private
289 }
290
291 // Returns the attributes from a
292 // match and an apool
293
294 * @param {{attrs: string,
295     operator: string, value:
296     string}} match - A
297     _parseMatch
298 * @param {{numToAttrib: array}}
299     apool - An attribute pool
300     from an
301     Etherpad server
302 * @returns {*}
303 * @private
304 */
305 this._getAttributesFromMatch =
306     function (match, apool) {
307         var i;
308         if (!match.attrs.length) return
309         [];
310         i = parseInt(match.attrs.substr
311             (1));
312         return [apool.numToAttrib[i]]
313     };
314
315 // Reads the charbank from a
316 // changeset string
317
318 * @param {string} str - A
319     serialized changeset
320 * @returns {string|null} - The
321     charbank or null
322 * if there is no charbank
323 * @private
324 }
325
326 // Returns the attributes from a
327 // match and an apool
328
329 * @param {{attrs: string,
330     operator: string, value:
331     string}} match - A
332     _parseMatch
333 * @param {{numToAttrib: array}}
334     apool - An attribute pool
335     from an
336     Etherpad server
337 * @returns {*}
338 * @private
339 */
340 this._getAttributesFromMatch =
341     function (match, apool) {
342         var i;
343         if (!match.attrs.length) return
344         [];
345         i = parseInt(match.attrs.substr
346             (1));
347         return [apool.numToAttrib[i]]
348     };
349
350 // Reads the charbank from a
351 // changeset string
352
353 * @param {string} str - A
354     serialized changeset
355 * @returns {string|null} - The
356     charbank or null
357 * if there is no charbank
358 * @private
359 }
360
361 // Returns the attributes from a
362 // match and an apool
363
364 * @param {{attrs: string,
365     operator: string, value:
366     string}} match - A
367     _parseMatch
368 * @param {{numToAttrib: array}}
369     apool - An attribute pool
370     from an
371     Etherpad server
372 * @returns {*}
373 * @private
374 */
375 this._getAttributesFromMatch =
376     function (match, apool) {
377         var i;
378         if (!match.attrs.length) return
379         [];
380         i = parseInt(match.attrs.substr
381             (1));
382         return [apool.numToAttrib[i]]
383     };
384
385 // Reads the charbank from a
386 // changeset string
387
388 * @param {string} str - A
389     serialized changeset
390 * @returns {string|null} - The
391     charbank or null
392 * if there is no charbank
393 * @private
394 }
395
396 // Returns the attributes from a
397 // match and an apool
398
399 * @param {{attrs: string,
400     operator: string, value:
401     string}} match - A
402     _parseMatch
403 * @param {{numToAttrib: array}}
404     apool - An attribute pool
405     from an
406     Etherpad server
407 * @returns {*}
408 * @private
409 */
410 this._getAttributesFromMatch =
411     function (match, apool) {
412         var i;
413         if (!match.attrs.length) return
414         [];
415         i = parseInt(match.attrs.substr
416             (1));
417         return [apool.numToAttrib[i]]
418     };
419
420 // Reads the charbank from a
421 // changeset string
422
423 * @param {string} str - A
424     serialized changeset
425 * @returns {string|null} - The
426     charbank or null
427 * if there is no charbank
428 * @private
429 }
430
431 // Returns the attributes from a
432 // match and an apool
433
434 * @param {{attrs: string,
435     operator: string, value:
436     string}} match - A
437     _parseMatch
438 * @param {{numToAttrib: array}}
439     apool - An attribute pool
440     from an
441     Etherpad server
442 * @returns {*}
443 * @private
444 */
445 this._getAttributesFromMatch =
446     function (match, apool) {
447         var i;
448         if (!match.attrs.length) return
449         [];
450         i = parseInt(match.attrs.substr
451             (1));
452         return [apool.numToAttrib[i]]
453     };
454
455 // Reads the charbank from a
456 // changeset string
457
458 * @param {string} str - A
459     serialized changeset
460 * @returns {string|null} - The
461     charbank or null
462 * if there is no charbank
463 * @private
464 }
465
466 // Returns the attributes from a
467 // match and an apool
468
469 * @param {{attrs: string,
470     operator: string, value:
471     string}} match - A
472     _parseMatch
473 * @param {{numToAttrib: array}}
474     apool - An attribute pool
475     from an
476     Etherpad server
477 * @returns {*}
478 * @private
479 */
480 this._getAttributesFromMatch =
481     function (match, apool) {
482         var i;
483         if (!match.attrs.length) return
484         [];
485         i = parseInt(match.attrs.substr
486             (1));
487         return [apool.numToAttrib[i]]
488     };
489
490 // Reads the charbank from a
491 // changeset string
492
493 * @param {string} str - A
494     serialized changeset
495 * @returns {string|null} - The
496     charbank or null
497 * if there is no charbank
498 * @private
499 }
500
501 // Returns the attributes from a
502 // match and an apool
503
504 * @param {{attrs: string,
505     operator: string, value:
506     string}} match - A
507     _parseMatch
508 * @param {{numToAttrib: array}}
509     apool - An attribute pool
510     from an
511     Etherpad server
512 * @returns {*}
513 * @private
514 */
515 this._getAttributesFromMatch =
516     function (match, apool) {
517         var i;
518         if (!match.attrs.length) return
519         [];
520         i = parseInt(match.attrs.substr
521             (1));
522         return [apool.numToAttrib[i]]
523     };
524
525 // Reads the charbank from a
526 // changeset string
527
528 * @param {string} str - A
529     serialized changeset
530 * @returns {string|null} - The
531     charbank or null
532 * if there is no charbank
533 * @private
534 }
535
536 // Returns the attributes from a
537 // match and an apool
538
539 * @param {{attrs: string,
540     operator: string, value:
541     string}} match - A
542     _parseMatch
543 * @param {{numToAttrib: array}}
544     apool - An attribute pool
545     from an
546     Etherpad server
547 * @returns {*}
548 * @private
549 */
550 this._getAttributesFromMatch =
551     function (match, apool) {
552         var i;
553         if (!match.attrs.length) return
554         [];
555         i = parseInt(match.attrs.substr
556             (1));
557         return [apool.numToAttrib[i]]
558     };
559
560 // Reads the charbank from a
561 // changeset string
562
563 * @param {string} str - A
564     serialized changeset
565 * @returns {string|null} - The
566     charbank or null
567 * if there is no charbank
568 * @private
569 }
570
571 // Returns the attributes from a
572 // match and an apool
573
574 * @param {{attrs: string,
575     operator: string, value:
576     string}} match - A
577     _parseMatch
578 * @param {{numToAttrib: array}}
579     apool - An attribute pool
580     from an
581     Etherpad server
582 * @returns {*}
583 * @private
584 */
585 this._getAttributesFromMatch =
586     function (match, apool) {
587         var i;
588         if (!match.attrs.length) return
589         [];
590         i = parseInt(match.attrs.substr
591             (1));
592         return [apool.numToAttrib[i]]
593     };
594
595 // Reads the charbank from a
596 // changeset string
597
598 * @param {string} str - A
599     serialized changeset
600 * @returns {string|null} - The
601     charbank or null
602 * if there is no charbank
603 * @private
604 }
605
606 // Returns the attributes from a
607 // match and an apool
608
609 * @param {{attrs: string,
610     operator: string, value:
611     string}} match - A
612     _parseMatch
613 * @param {{numToAttrib: array}}
614     apool - An attribute pool
615     from an
616     Etherpad server
617 * @returns {*}
618 * @private
619 */
620 this._getAttributesFromMatch =
621     function (match, apool) {
622         var i;
623         if (!match.attrs.length) return
624         [];
625         i = parseInt(match.attrs.substr
626             (1));
627         return [apool.numToAttrib[i]]
628     };
629
630 // Reads the charbank from a
631 // changeset string
632
633 * @param {string} str - A
634     serialized changeset
635 * @returns {string|null} - The
636     charbank or null
637 * if there is no charbank
638 * @private
639 }
640
641 // Returns the attributes from a
642 // match and an apool
643
644 * @param {{attrs: string,
645     operator: string, value:
646     string}} match - A
647     _parseMatch
648 * @param {{numToAttrib: array}}
649     apool - An attribute pool
650     from an
651     Etherpad server
652 * @returns {*}
653 * @private
654 */
655 this._getAttributesFromMatch =
656     function (match, apool) {
657         var i;
658         if (!match.attrs.length) return
659         [];
660         i = parseInt(match.attrs.substr
661             (1));
662         return [apool.numToAttrib[i]]
663     };
664
665 // Reads the charbank from a
666 // changeset string
667
668 * @param {string} str - A
669     serialized changeset
670 * @returns {string|null} - The
671     charbank or null
672 * if there is no charbank
673 * @private
674 }
675
676 // Returns the attributes from a
677 // match and an apool
678
679 * @param {{attrs: string,
680     operator: string, value:
681     string}} match - A
682     _parseMatch
683 * @param {{numToAttrib: array}}
684     apool - An attribute pool
685     from an
686     Etherpad server
687 * @returns {*}
688 * @private
689 */
690 this._getAttributesFromMatch =
691     function (match, apool) {
692         var i;
693         if (!match.attrs.length) return
694         [];
695         i = parseInt(match.attrs.substr
696             (1));
697         return [apool.numToAttrib[i]]
698     };
699
700 // Reads the charbank from a
701 // changeset string
702
703 * @param {string} str - A
704     serialized changeset
705 * @returns {string|null} - The
706     charbank or null
707 * if there is no charbank
708 * @private
709 }
710
711 // Returns the attributes from a
712 // match and an apool
713
714 * @param {{attrs: string,
715     operator: string, value:
716     string}} match - A
717     _parseMatch
718 * @param {{numToAttrib: array}}
719     apool - An attribute pool
720     from an
721     Etherpad server
722 * @returns {*}
723 * @private
724 */
725 this._getAttributesFromMatch =
726     function (match, apool) {
727         var i;
728         if (!match.attrs.length) return
729         [];
730         i = parseInt(match.attrs.substr
731             (1));
732         return [apool.numToAttrib[i]]
733     };
734
735 // Reads the charbank from a
736 // changeset string
737
738 * @param {string} str - A
739     serialized changeset
740 * @returns {string|null} - The
741     charbank or null
742 * if there is no charbank
743 * @private
744 }
745
746 // Returns the attributes from a
747 // match and an apool
748
749 * @param {{attrs: string,
750     operator: string, value:
751     string}} match - A
752     _parseMatch
753 * @param {{numToAttrib: array}}
754     apool - An attribute pool
755     from an
756     Etherpad server
757 * @returns {*}
758 * @private
759 */
760 this._getAttributesFromMatch =
761     function (match, apool) {
762         var i;
763         if (!match.attrs.length) return
764         [];
765         i = parseInt(match.attrs.substr
766             (1));
767         return [apool.numToAttrib[i]]
768     };
769
770 // Reads the charbank from a
771 // changeset string
772
773 * @param {string} str - A
774     serialized changeset
775 * @returns {string|null} - The
776     charbank or null
777 * if there is no charbank
778 * @private
779 }
780
781 // Returns the attributes from a
782 // match and an apool
783
784 * @param {{attrs: string,
785     operator: string, value:
786     string}} match - A
787     _parseMatch
788 * @param {{numToAttrib: array}}
789     apool - An attribute pool
790     from an
791     Etherpad server
792 * @returns {*}
793 * @private
794 */
795 this._getAttributesFromMatch =
796     function (match, apool) {
797         var i;
798         if (!match.attrs.length) return
799         [];
800         i = parseInt(match.attrs.substr
801             (1));
802         return [apool.numToAttrib[i]]
803     };
804
805 // Reads the charbank from a
806 // changeset string
807
808 * @param {string} str - A
809     serialized changeset
810 * @returns {string|null} - The
811     charbank or null
812 * if there is no charbank
813 * @private
814 }
815
816 // Returns the attributes from a
817 // match and an apool
818
819 * @param {{attrs: string,
820     operator: string, value:
821     string}} match - A
822     _parseMatch
823 * @param {{numToAttrib: array}}
824     apool - An attribute pool
825     from an
826     Etherpad server
827 * @returns {*}
828 * @private
829 */
830 this._getAttributesFromMatch =
831     function (match, apool) {
832         var i;
833         if (!match.attrs.length) return
834         [];
835         i = parseInt(match.attrs.substr
836             (1));
837         return [apool.numToAttrib[i]]
838     };
839
840 // Reads the charbank from a
841 // changeset string
842
843 * @param {string} str - A
844     serialized changeset
845 * @returns {string|null} - The
846     charbank or null
847 * if there is no charbank
848 * @private
849 }
850
851 // Returns the attributes from a
852 // match and an apool
853
854 * @param {{attrs: string,
855     operator: string, value:
856     string}} match - A
857     _parseMatch
858 * @param {{numToAttrib: array}}
859     apool - An attribute pool
860     from an
861     Etherpad server
862 * @returns {*}
863 * @private
864 */
865 this._getAttributesFromMatch =
866     function (match, apool) {
867         var i;
868         if (!match.attrs.length) return
869         [];
870         i = parseInt(match.attrs.substr
871             (1));
872         return [apool.numToAttrib[i]]
873     };
874
875 // Reads the charbank from a
876 // changeset string
877
878 * @param {string} str - A
879     serialized changeset
880 * @returns {string|null} - The
881     charbank or null
882 * if there is no charbank
883 * @private
884 }
885
886 // Returns the attributes from a
887 // match and an apool
888
889 * @param {{attrs: string,
890     operator: string, value:
891     string}} match - A
892     _parseMatch
893 * @param {{numToAttrib: array}}
894     apool - An attribute pool
895     from an
896     Etherpad server
897 * @returns {*}
898 * @private
899 */
900 this._getAttributesFromMatch =
901     function (match, apool) {
902         var i;
903         if (!match.attrs.length) return
904         [];
905         i = parseInt(match.attrs.substr
906             (1));
907         return [apool.numToAttrib[i]]
908     };
909
910 // Reads the charbank from a
911 // changeset string
912
913 * @param {string} str - A
914     serialized changeset
915 * @returns {string|null} - The
916     charbank or null
917 * if there is no charbank
918 * @private
919 }
920
921 // Returns the attributes from a
922 // match and an apool
923
924 * @param {{attrs: string,
925     operator: string, value:
926     string}} match - A
927     _parseMatch
928 * @param {{numToAttrib: array}}
929     apool - An attribute pool
930     from an
931     Etherpad server
932 * @returns {*}
933 * @private
934 */
935 this._getAttributesFromMatch =
936     function (match, apool) {
937         var i;
938         if (!match.attrs.length) return
939         [];
940         i = parseInt(match.attrs.substr
941             (1));
942         return [apool.numToAttrib[i]]
943     };
944
945 // Reads the charbank from a
946 // changeset string
947
948 * @param {string} str - A
949     serialized changeset
950 * @returns {string|null} - The
951     charbank or null
952 * if there is no charbank
953 * @private
954 }
955
956 // Returns the attributes from a
957 // match and an apool
958
959 * @param {{attrs: string,
960     operator: string, value:
961     string}} match - A
962     _parseMatch
963 * @param {{numToAttrib: array}}
964     apool - An attribute pool
965     from an
966     Etherpad server
967 * @returns {*}
968 * @private
969 */
970 this._getAttributesFromMatch =
971     function (match, apool) {
972         var i;
973         if (!match.attrs.length) return
974         [];
975         i = parseInt(match.attrs.substr
976             (1));
977         return [apool.numToAttrib[i]]
978     };
979
980 // Reads the charbank from a
981 // changeset string
982
983 * @param {string} str - A
984     serialized changeset
985 * @returns {string|null} - The
986     charbank or null
987 * if there is no charbank
988 * @private
989 }
990
991 // Returns the attributes from a
992 // match and an apool
993
994 * @param {{attrs: string,
995     operator: string, value:
996     string}} match - A
997     _parseMatch
998 * @param {{numToAttrib: array}}
999     apool - An attribute pool
1000     from an
1001     Etherpad server
1002 * @returns {*}
1003 * @private
1004 */
1005 this._getAttributesFromMatch =
1006     function (match, apool) {
1007         var i;
1008         if (!match.attrs.length) return
1009         [];
1010         i = parseInt(match.attrs.substr
1011             (1));
1012         return [apool.numToAttrib[i]]
1013     };
1014
1015 // Reads the charbank from a
1016 // changeset string
1017
1018 * @param {string} str - A
1019     serialized changeset
1020 * @returns {string|null} - The
1021     charbank or null
1022 * if there is no charbank
1023 * @private
1024 }
1025
1026 // Returns the attributes from a
1027 // match and an apool
1028
1029 * @param {{attrs: string,
1030     operator: string, value:
1031     string}} match - A
1032     _parseMatch
1033 * @param {{numToAttrib: array}}
1034     apool - An attribute pool
1035     from an
1036     Etherpad server
1037 * @returns {*}
1038 * @private
1039 */
1040 this._getAttributesFromMatch =
1041     function (match, apool) {
1042         var i;
1043         if (!match.attrs.length) return
1044         [];
1045         i = parseInt(match.attrs.substr
1046             (1));
1047         return [apool.numToAttrib[i]]
1048     };
1049
1050 // Reads the charbank from a
1051 // changeset string
1052
1053 * @param {string} str - A
1054     serialized changeset
1055 * @returns {string|null} - The
1056     charbank or null
1057 * if there is no charbank
1058 * @private
1059 }
1060
1061 // Returns the attributes from a
1062 // match and an apool
1063
1064 * @param {{attrs: string,
1065     operator: string, value:
1066     string}} match - A
1067     _parseMatch
1068 * @param {{numToAttrib: array}}
1069     apool - An attribute pool
1070     from an
1071     Etherpad server
1072 * @returns {*}
1073 * @private
1074 */
1075 this._getAttributesFromMatch =
1076     function (match, apool) {
1077         var i;
1078         if (!match.attrs.length) return
1079         [];
1080         i = parseInt(match.attrs.substr
1081             (1));
1082         return [apool.numToAttrib[i]]
1083     };
1084
1085 // Reads the charbank from a
1086 // changeset string
1087
1088 * @param {string} str - A
1089     serialized changeset
1090 * @returns {string|null} - The
1091     charbank or null
1092 * if there is no charbank
1093 * @private
1094 }
1095
1096 // Returns the attributes from a
1097 // match and an apool
1098
1099 * @param {{attrs: string,
1100     operator: string, value:
1101     string}} match - A
1102     _parseMatch
1103 * @param {{numToAttrib: array}}
1104     apool - An attribute pool
1105     from an
1106     Etherpad server
1107 * @returns {*}
1108 * @private
1109 */
1110 this._getAttributesFromMatch =
1111     function (match, apool) {
1112         var i;
1113         if (!match.attrs.length) return
1114         [];
1115         i = parseInt(match.attrs.substr
1116             (1));
1117         return [apool.numToAttrib[i]]
1118     };
1119
1120 // Reads the charbank from a
1121 // changeset string
1122
1123 * @param {string} str - A
1124     serialized changeset
1125 * @returns {string|null} - The
1126     charbank or null
1127 * if there is no charbank
1128 * @private
1129 }
1130
1131 // Returns the attributes from a
1132 // match and an apool
1133
1134 * @param {{attrs: string,
1135     operator: string, value:
1136     string}} match - A
1137     _parseMatch
1138 * @param {{numToAttrib: array}}
1139     apool - An attribute pool
1140     from an
1141     Etherpad server
1142 * @returns {*}
1143 * @private
1144 */
1145 this._getAttributesFromMatch =
1146     function (match, apool) {
1147         var i;
1148         if (!match.attrs.length) return
1149         [];
1150         i = parseInt(match.attrs.substr
1151             (1));
1152         return [apool.numToAttrib[i]]
1153     };
1154
1155 // Reads the charbank from a
1156 // changeset string
1157
1158 * @param {string} str - A
1159     serialized changeset
1160 * @returns {string|null} - The
1161     charbank or null
1162 * if there is no charbank
1163 * @private
1164 }
1165
1166 // Returns the attributes from a
1167 // match and an apool
1168
1169 * @param {{attrs: string,
1170     operator: string, value:
1171     string}} match - A
1172     _parseMatch
1173 * @param {{numToAttrib: array}}
1174     apool - An attribute pool
1175     from an
1176     Etherpad server
1177 * @returns {*}
1178 * @private
1179 */
1180 this._getAttributesFromMatch =
1181     function (match, apool) {
1182         var i;
1183         if (!match.attrs.length) return
1184         [];
1185         i = parseInt(match.attrs.substr
1186             (1));
1187         return [apool.numToAttrib[i]]
1188     };
1189
1190 // Reads the charbank from a
1191 // changeset string
1192
1193 * @param {string} str - A
1194     serialized changeset
1195 * @returns {string|null} - The
1196     charbank or null
1197 * if there is no charbank
1198 * @private
1199 }
1200
1201 // Returns the attributes from a
1202 // match and an apool
1203
1204 * @param {{attrs: string,
1205     operator: string, value:
1206     string}} match - A
1207     _parseMatch
1208 * @param {{numToAttrib: array}}
1209     apool - An attribute pool
1210     from an
1211     Etherpad server
1212 * @returns {*}
1213 * @private
1214 */
1215 this._getAttributesFromMatch =
1216     function (match, apool) {
1217         var i;
1218         if (!match.attrs.length) return
1219         [];
1220         i = parseInt(match.attrs.substr
1221             (1));
1222         return [apool.numToAttrib[i]]
1223     };
1224
1225 // Reads the charbank from a
1226 // changeset string
1227
1228 * @param {string} str - A
1229     serialized changeset
1230 * @returns {string|null} - The
1231     charbank or null
1232 * if there is no charbank
1233 * @private
1234 }
1235
1236 // Returns the attributes from a
1237 // match and an apool
1238
1239 * @param {{attrs: string,
1240     operator: string, value:
1241     string}} match - A
1242     _parseMatch
1243 * @param {{numToAttrib: array}}
1244     apool - An attribute pool
1245     from an
1246     Etherpad server
1247 * @returns {*}
1248 * @private
1249 */
1250 this._getAttributesFromMatch =
1251     function (match, apool) {
1252         var i;
1253         if (!match.attrs.length) return
1254         [];
1255         i = parseInt(match.attrs.substr
1256             (1));
1257         return [apool.numToAttrib[i]]
1258     };
1259
1260 // Reads the charbank from a
1261 // changeset string
1262
1263 * @param {string} str - A
1264     serialized changeset
1265 * @returns {string|null} - The
1266     charbank or null
1267 * if there is no charbank
1268 * @private
1269 }
1270
1271 // Returns the attributes from a
1272 // match and an apool
1273
1274 * @param {{attrs: string,
1275     operator: string, value:
1276     string}} match - A
1277     _parseMatch
1278 * @param {{numToAttrib: array}}
1279     apool - An attribute pool
1280     from an
1281     Etherpad server
1282 * @returns {*}
1283 * @private
1284 */
1285 this._getAttributesFromMatch =
1286     function (match, apool) {
1287         var i;
1288         if (!match.attrs.length) return
1289         [];
1290         i = parseInt(match.attrs.substr
1291             (1));
1292         return [apool.numToAttrib[i]]
1293     };
1294
1295 // Reads the charbank from a
1296 // changeset string
1297
1298 * @param {string} str - A
1299     serialized changeset
1300 * @returns {string|null} - The
1301     charbank or null
1302 * if there is no charbank
1303 * @private
1304 }
1305
1306 // Returns the attributes from a
1307 // match and an apool
1308
1309 * @param {{attrs: string,
1310     operator: string, value:
1311     string}} match - A
1312     _parseMatch
1313 * @param {{numToAttrib: array}}
1314     apool - An attribute pool
1315     from an
1316     Etherpad server
1317 * @returns {*}
1318 * @private
1319 */
1320 this._getAttributesFromMatch =
1321     function (match, apool) {
1322         var i;
1323         if (!match.attrs.length) return
1324         [];
1325         i = parseInt(match.attrs.substr
1326             (1));
1327         return [apool.numToAttrib[i]]
1328     };
1329
1330 // Reads the charbank from a
1331 // changeset string
1332
1333 * @param {string} str - A
1334     serialized changeset
1335 * @returns {string|null} - The
1336     charbank or null
1337 * if there is no charbank
1338 * @private
1339 }
1340
1341 // Returns the attributes from a
1342 // match and an apool
1343
1344 * @param {{attrs: string,
1345     operator: string, value:
1346     string}} match - A
1347     _parseMatch
1348 * @param {{numToAttrib: array}}
1349     apool - An attribute pool
1350     from an
1351     Etherpad server
1352 * @returns {*}
1353 * @private
1354 */
1355 this._getAttributesFromMatch =
1356     function (match, apool) {
1357         var i;
1358         if (!match.attrs.length) return
1359         [];
1360         i = parseInt(match.attrs.substr
1361             (1));
1362         return [apool.numToAttrib[i]]
1363     };
1364
1365 // Reads the charbank from a
1366 // changeset string
1367
1368 * @param {string} str - A
1369     serialized changeset
1370 * @returns {string|null} - The
1371     charbank or null
1372 * if there is no charbank
1373 * @private
1374 }
1375
1376 // Returns the attributes from a
1377 // match and an apool
1378
1379 * @param {{attrs: string,
1380     operator: string, value:
1381     string}} match - A
1382     _parseMatch
1383 * @param {{numToAttrib: array}}
1384     apool - An attribute pool
1385     from an
1386     Etherpad server
1387 * @returns {*}
1388 * @private
1389 */
1390 this._getAttributesFromMatch =
1391     function (match, apool) {
1392         var i;
1393         if (!match.attrs.length) return
1394         [];
1395         i = parseInt(match.attrs.substr
1396             (1));
1397         return [apool.numToAttrib[i]]
1398     };
1399
1400 // Reads the charbank from a
1401 // changeset string
1402
1403 * @param {string} str - A
1404     serialized changeset
1405 * @returns {string|null} - The
1406     charbank or null
1407 * if there is no charbank
1408 * @private
1409 }
1410
1411 // Returns the attributes from a
1412 // match and an apool
1413
1414 * @param {{attrs: string,
1415     operator: string, value:
1416     string}} match - A
1417     _parseMatch
1418 * @param {{numToAttrib: array}}
1419     apool - An attribute pool
1420     from an
1421     Etherpad server
1422 * @returns {*}
1423 * @private
1424 */
1425 this._getAttributesFromMatch =
1426     function (match, apool) {
1427         var i;
1428         if (!match.attrs.length) return
1429         [];
1430         i = parseInt(match.attrs.substr
1431             (1));
1432         return [apool.numToAttrib[i]]
1433     };
1434
1435 // Reads the charbank from a
1436 // changeset string
1437
1438 * @param {string} str - A
1439     serialized changeset
1440 * @returns {string|null} - The
1441     charbank or null
1442 * if there is no charbank
1443 * @private
1444 }
1445
1446 // Returns the attributes from a
1447 // match and an apool
1448
1449 * @param {{attrs: string,
1450     operator: string, value:
1451     string}} match - A
1452     _parseMatch
1453 * @param {{numToAttrib: array}}
1454     apool - An attribute pool
1455     from an
1456     Etherpad server
1457 * @returns {*}
1458 * @private
1459 */
1460 this._getAttributesFromMatch =
1461     function (match, apool) {
1462         var i;
1463         if (!match.attrs.length) return
1464         [];
1465         i = parseInt(match.attrs.substr
1466             (1));
1467         return [apool.numToAttrib[i]]
1468     };
1469
1470 // Reads the charbank from a
1471 // changeset string
1472
1473 * @param {string} str - A
1474     serialized changeset
1475 * @returns {string|null} - The
1476     charbank or null
1477 * if there is no charbank
1478 * @private
1479 }
1480
1481 // Returns the attributes from a
1482 // match and an apool
1483
1484 * @param {{attrs: string,
1485     operator: string, value:
1486     string}} match - A
1487     _parseMatch
1488 * @param {{numToAttrib: array}}
1489     apool - An attribute pool
1490     from an
1491     Etherpad server
1492 * @returns {*}
1493 * @private
1494 */
1495 this._getAttributesFromMatch =
1496     function (match, apool) {
1497         var i;
1498         if (!match.attrs.length) return
1499         [];
1500         i = parseInt(match.attrs.substr
1501             (1));
1502         return [apool.numToAttrib[i]]
15
```


Changeset ;
 Listing 37:
 /plugins/etherpad/changeset.js

```

1  'use strict';
2
3  var Type = require(' ../../.../core');
4
5  /**
6   * Creates a new Type.Etherpad.
7   *   Changeset instance
8   * @constructor
9   */
10 Type.Etherpad.Changeset.Changes.
11   Change = function () {
12
13 };
14 (function () {
15   this.apply = function (content,
16     localCaret) {

```

```

36   this.getLength = function () {
37     return 0;
38   };
39
40   }.call(Type.Etherpad.Changeset.
41     Changes.Change.prototype);
42
43   Type.Etherpad.Changeset.Changes.
44     Change.fromMatch = function (
45       match) {
46     return new Type.Etherpad.Changeset
47       .Changes.Change();
48
49     module.exports = Type.Etherpad.
50       Changeset.Changes.Change;

```

Listing 38: /plugins/etherpad/changes/change.js

```

1  'use strict';
2
3  var Type = require(' ../../.../core');
4
5  /**
6   * Creates a new Type.Etherpad.
7   *   Changeset instance
8   * @param {number} delta - The
9   *   relative movement
10   * @param {number} [absolute] - The
11   *   absolute text position
12   * @constructor
13
14 Type.Etherpad.Changeset.Changes.
15   Movement = function (delta,

```

```

24   absolute) {
25     this.delta = delta;
26     this.absolute = absolute || null;
27   };
28
29   /** Inherit from Etherpad change
30   */
31   Type.OOP.inherits(Type.Etherpad.
32     Changeset.Changes.Movement,
33     Type.Etherpad.Changeset.Changes
34       .Change);
35
36   (function () {

```

```

24   /**
25    * Etherpad's serialized string
26    *   for this operation
27    * @type {string}
28    */
29   this.op = '';
30
31   }.call(Type.Etherpad.Changeset.
32     Changes.Movement.prototype);
33
34   /**
35    * @param match
36    * @returns {Type.Etherpad.Changeset

```

```

37     .Changes.Movement}
38     * @constructor
39     Type.Etherpad.Changeset.Changes.
40     Movement.fromMatch = function (
41     match) {
42     return new Type.Etherpad.Changeset
43     .Changes.Movement(parseInt(
44     match.value, 36));
45
46     }
47     /**
48     * Creates a new Movement instance
49
50     from an object containing the
51     delta
52     * and the absolute text offset.
53     * @param {{ absolute: number, stack
54     : number[] }} offset - An
55     object
56     containing offset information
57
58     Type.Etherpad.Changeset.Changes.
59     Movement.fromOffsetObject =
60     function (offset, match) {
61     var delta = offset.stack[offset.
62     stack.length - 1];
63
64     (function () {
65
66     /**
67     * Etherpad's serialized string
68     for this operation
69     * @type {string}
70
71     this.op = '+';
72
73     /**
74     * @returns {string}
75
76     this.getOperation = function () {
77     return this.op + this.text.
78     length;
79
80     };
81
82     /**
83     * @returns {number}
84
85     this.getLength = function () {
86     return this.length;
87
88     };
89
90     /**
91     * @param {Type.Content} content -
92
93     The content this changeset
94     should be applied to
95     * @param {Type.Caret} localCaret
96     - The local user's caret
97     * @returns {Type.Etherpad.
98     Changeset.Changes.Insertion}
99     - This instance
100
101     this.apply = function (content,
102     localCaret) {
103     content.insert(this.start, this.
104     text);
105     localCaret.moveBy(this.length);
106     return this;
107
108     /**
109     * @param {Type.Etherpad.Changeset
110     .Changes.Insertion} that -
111     instance
112     * @returns {boolean}
113
114     this.mergable = function (that) {
115     return that instanceof Type.
116     Etherpad.Changeset.Changes.
117     Insertion &&

```

Listing 39: /plugins/etherpad/changes/movement.js

```

67     this.start <= that.start &&
68     that.start <= this.end;
69     };
70     /**
71     * @param {Type.Etherpad.Changeset
72     * -Changes.Insertion} that -
73     * Another Insertion
74     * instance
75     * @returns {Type.Etherpad.
76     * Changeset.Changes.Insertion}
77     - This instance
78     */
79     this.merge = function (that) {
80     var offset = that.start - this.
81     start;
82     this.text = this.text.substr(0,
83     offset) + that.text + this.
84     text.substr(offset);
85     return this;
86 }
87
88 for this operation
89 * @type {string}
90 */
91 this.op = '-';
92 /**
93 * @returns {string}
94 */
95 this.getOperation = function () {
96     return this.op + this.length;
97 };
98 /**
99 * @returns {number}
100 */
101 this.getLength = function () {
102     return this.length * -1;
103 };
104 /**
105 * @param {Type.Content} content -
106 * The content this changeset
107 * should be applied to
108 * @param {Type.Caret} localCaret
109 * - The local user's caret
110 * @returns {Type.Etherpad.
111 * Changeset.Changes.Insertion}
112 * - This instance
113 */
114
115 use strict';
116 var Type = require('..../core');
117 /**
118 * Creates a new Type.Etherpad.
119 * Changeset instance
120 * @constructor
121 */
122 Type.Etherpad.Changeset.Changes.
123 Removal = function (offset,
124 length) {
125     this.start = offset;
126     this.length = length;
127     this.end = offset + length;
128 };
129 /**
130 * Inherit from Etherpad change
131 */
132 Type.OOP.inherits(Type.Etherpad.
133 Changeset.Changes.Removal, Type.
134 Etherpad.Changeset.Changes.
135 Change);
136
137 (function () {
138 /**
139 * Etherpad's serialized string

```

```

73   .Changes.Removal(offset,
74   absolute, parseInt(match.
    value, 36));
75 };

1  'use strict';
2  var Type = require('.../.../core');
3  /**
4   * Creates a new Type.Etherpad.
5   * Changeset instance
6   *
7   * @constructor
8   */
9  Type.Etherpad.Changeset.Changes.
10  Formatting = function (command,
    offset, length, remove) {
11    this.command = command;
12    this.start = offset;
13    this.length = length;
14    this.end = offset + length;
15    this.remove = !!remove;
16  };
17  /**
18   * Inherit from Etherpad change
19   */
20  Type.OOP.inherits(Type.Etherpad.
21  Changeset.Changes.Formatting,
    Type.Etherpad.Changeset.Changes.
    Change);
22
23  (function () {
24
25    /**
26     * Etherpad's serialized string
27     for this operation
    Changeset.Changes.Removal;
    Listing 41: /plugins/ether-
    pad/changes/removal.js

    module.exports = Type.Etherpad.
    pad/changes/removal.js

    * @type {string}
    */
    this.op = '=';
    /**
    * Maps Etherpad commands to tags
    to apply in the editor
    */
    * @type {{bold: string}}
    * @private
    */
    this._tagMap = {
      bold : 'strong',
      italic : 'em',
      underline : 'u',
      strikethrough : 's',
    };
    /**
    * @param {Type.Content} content -
    The content this changeset
    should be applied to
    * @param {Type.Caret} [localCaret]
    - The local user's caret
    * @returns {Type.Etherpad.
    Changeset.Changes.Insertion}
    - This instance
    */
    this.apply = function (content,
      localCaret) {
      if (this.command !== 'author') {
        content.format(this._tagMap[
    Changeset.Changes.Removal;
    Listing 42: /plugins/ether-
    pad/changes/formatting.js

    module.exports = Type.Etherpad.
    Changeset.Changes.Formatting;
    Listing 42: /plugins/ether-
    pad/changes/formatting.js

```

```

1  2 'use strict';
2  3
3  4 var Type = require('.../core');
4  5
5  6 /**
6  7  * Creates a new Type.Etherpad.
7  8  * ChangesetSerializer instance
8  9  *
9  10 * This class can be used to
10 11 * serialize a Type.Etherpad.
11 12 * Changeset
12 13 * server
13 14 *
14 15 * @param {Type.Etherpad.Changeset}
15 16 * changeset
16 17 * @constructor
17 18
18 19 Type.Etherpad.ChangesetSerializer =
19 20 function (changeset) {
20 21   this._operations = this.
21 22   _getOperations(changeset);
22 23
23 24   (function () {
24 25     /**
25 26     * Returns a serialized changeset
26 27     * string based on the length
27 28     * of
28 29     * a given string or the text
29 30     * contents of a given element
30 31     *
31 32     * @param {string|Element} base -
32 33     * Either a string or an
33 34     * element
34 35     * @returns {string} - The
35 36     * changeset string
36 37     */
37 38     this.getString = function (base) {
38 39       var changeset, len, i;
39 40       len = this._operations.length;
40 41
41 42       changeset = this.
42 43       _baseLengthString(base);
43 44       changeset += this.
44 45       _lengthChangeString();
45 46       changeset += this.
46 47       _operationsString(this.
47 48       _operations[0], null, base)
48 49       ;
49 50       for (i = 1; i < len; i += 1) {
50 51         changeset += this.
51 52         _operationString(this.
52 53         _operations[i], this.
53 54         _operations[i - 1]);
54 55       }
55 56       changeset += this.
56 57       _charbankString();
57 58       return changeset;
58 59     };
59 60     /**
60 61     * Returns the length parameter
61 62     * for the changeset string
62 63     */
63 64     Returns the length of either a
64 65     string or the text inside
65 66     an element as a 36 base encoded
66 67     number, prepended by 'Z.'
67 68     */
68 69     @param {string|Element} base -
69 70     Either a string or an
70 71     element
71 72     @returns {string} - The 36 base
72 73     encoded number
73 74     */
74 75     @private
75 76     this._baseLengthString = function
76 77     (base) {
77 78       return 'Z.' + this._lengthFor(
78 79       base).toString(36);
79 80     };
80 81
81 82 len = this._operations.length;
82 83
83 84 changeset = this.
84 85 _baseLengthString(base);
85 86 changeset += this.
86 87 _lengthChangeString();
87 88 changeset += this.
88 89 _operationsString(this.
89 90 _operations[0], null, base)
90 91 ;
91 92 for (i = 1; i < len; i += 1) {
92 93   changeset += this.
93 94   _operationString(this.
94 95   _operations[i], this.
95 96   _operations[i - 1]);
96 97 }
97 98 changeset += this.
98 99 _charbankString();
99 100 return changeset;
100 101 };
101 102 /**
102 103 * Returns a serialized operation
103 104 as a string
104 105 */
105 106 @param {{op: string, start:
106 107 number, end: number, text:
107 108 string}}{op: string, start:
108 109 number, numChars: number}}
109 110 operation
110 111 An insertion or removal
111 112 object
112 113 @param {{op: string, start:
113 114 number, end: number, text:
114 115 string}}{op: string, start:
115 116 number, numChars: number}} |
116 117 prev]
117 118 The operation before this
118 119 operation
119 120 @returns {string} - The
120 121 serialized string for the
121 122 operation
122 123 @private
123 124 */
124 125 this._operationString = function (
125 126 operation, prev, base) {
126 127   var offset, hack, operatorSnd;
127 128
128 129   offset = Math.abs(count).toString();
129 130   count = this.
130 131   _lengthChangeString =
131 132   function () {
132 133     return (count > 0 ? '>' : '<') +
133 134     Math.abs(count).toString();
134 135   };
135 136   Returns the parameter for the
136 137   changeset string that
137 138   determines
138 139   the change in the length of the
139 140   text.
140 141   */
141 142   @returns {string}
142 143   @private
143 144   */
144 145   this._lengthChangeString =
145 146   function () {
146 147     return (count > 0 ? '>' : '<') +
147 148     Math.abs(count).toString();
148 149   };
149 150   Returns a serialized operation
150 151   as a string
151 152   */
152 153   @param {{op: string, start:
153 154   number, end: number, text:
154 155   string}}{op: string, start:
155 156   number, numChars: number}}
156 157   operation
157 158   An insertion or removal
158 159   object
159 160   @param {{op: string, start:
160 161   number, end: number, text:
161 162   string}}{op: string, start:
162 163   number, numChars: number}} |
163 164   prev]
164 165   The operation before this
165 166   operation
166 167   @returns {string} - The
167 168   serialized string for the
168 169   operation
169 170   @private
170 171   */
171 172   this._operationString = function (
172 173   operation, prev, base) {
173 174     var offset, hack, operatorSnd;
174 175
175 176     offset = Math.abs(count).toString();
176 177     count = this.
177 178     _lengthChangeString =
178 179     function () {
179 180       return (count > 0 ? '>' : '<') +
179 181       Math.abs(count).toString();
180 182     };
182 183     Returns a serialized operation
183 184     as a string
184 185     */
185 186     @param {{op: string, start:
186 187     number, end: number, text:
187 188     string}}{op: string, start:
188 189     number, numChars: number}}
189 190     operation
189 191     An insertion or removal
190 192     object
191 193     @param {{op: string, start:
193 194     number, end: number, text:
194 195     string}}{op: string, start:
195 196     number, numChars: number}} |
196 197     prev]
197 198     The operation before this
198 199     operation
199 200     @returns {string} - The
200 201     serialized string for the
201 202     operation
202 203     @private
203 204     */
204 205     this._operationString = function (
205 206     operation, prev, base) {
206 207       var offset, hack, operatorSnd;
207 208
208 209       offset = Math.abs(count).toString();
209 210       count = this.
210 211       _lengthChangeString =
211 212       function () {
212 213         return (count > 0 ? '>' : '<') +
212 214         Math.abs(count).toString();
213 215       };
215 216       Returns a serialized operation
216 217       as a string
217 218       */
218 219       @param {{op: string, start:
219 220       number, end: number, text:
220 221       string}}{op: string, start:
221 222       number, numChars: number}}
222 223       operation
223 224       An insertion or removal
224 225       object
225 226       @param {{op: string, start:
226 227       number, end: number, text:
227 228       string}}{op: string, start:
228 229       number, numChars: number}} |
229 230       prev]
230 231       The operation before this
231 232       operation
232 233       @returns {string} - The
233 234       serialized string for the
234 235       operation
235 236       @private
236 237       */
237 238       this._operationString = function (
238 239       operation, prev, base) {
239 240         var offset, hack, operatorSnd;
240 241
241 242         offset = Math.abs(count).toString();
242 243         count = this.
243 244         _lengthChangeString =
244 245         function () {
245 246           return (count > 0 ? '>' : '<') +
245 247           Math.abs(count).toString();
246 248         };
248 249         Returns a serialized operation
249 250         as a string
250 251         */
251 252         @param {{op: string, start:
252 253         number, end: number, text:
253 254         string}}{op: string, start:
254 255         number, numChars: number}}
255 256         operation
256 257         An insertion or removal
257 258         object
258 259         @param {{op: string, start:
259 260         number, end: number, text:
260 261         string}}{op: string, start:
261 262         number, numChars: number}} |
262 263         prev]
263 264         The operation before this
264 265         operation
265 266         @returns {string} - The
266 267         serialized string for the
267 268         operation
268 269         @private
269 270         */
270 271         this._operationString = function (
271 272         operation, prev, base) {
272 273           var offset, hack, operatorSnd;
273 274
274 275           offset = Math.abs(count).toString();
275 276           count = this.
276 277           _lengthChangeString =
277 278           function () {
278 279             return (count > 0 ? '>' : '<') +
278 280             Math.abs(count).toString();
279 281           };
281 282           Returns a serialized operation
282 283           as a string
283 284           */
284 285           @param {{op: string, start:
285 286           number, end: number, text:
286 287           string}}{op: string, start:
287 288           number, numChars: number}}
288 289           operation
289 290           An insertion or removal
290 291           object
291 292           @param {{op: string, start:
292 293           number, end: number, text:
293 294           string}}{op: string, start:
294 295           number, numChars: number}} |
295 296           prev]
296 297           The operation before this
297 298           operation
298 299           @returns {string} - The
299 300           serialized string for the
299 301           operation
300 302           @private
301 303           */
303 304           this._operationString = function (
304 305           operation, prev, base) {
305 306             var offset, hack, operatorSnd;
306 307
307 308             offset = Math.abs(count).toString();
308 309             count = this.
309 310             _lengthChangeString =
310 311             function () {
311 312               return (count > 0 ? '>' : '<') +
311 313               Math.abs(count).toString();
312 314             };
314 315             Returns a serialized operation
315 316             as a string
316 317             */
317 318             @param {{op: string, start:
318 319             number, end: number, text:
319 320             string}}{op: string, start:
320 321             number, numChars: number}}
321 322             operation
322 323             An insertion or removal
323 324             object
324 325             @param {{op: string, start:
325 326             number, end: number, text:
326 327             string}}{op: string, start:
327 328             number, numChars: number}} |
328 329             prev]
329 330             The operation before this
330 331             operation
331 332             @returns {string} - The
332 333             serialized string for the
333 334             operation
334 335             @private
335 336             */
336 337             this._operationString = function (
337 338             operation, prev, base) {
338 339               var offset, hack, operatorSnd;
339 340
340 341               offset = Math.abs(count).toString();
341 342               count = this.
342 343               _lengthChangeString =
343 344               function () {
344 345                 return (count > 0 ? '>' : '<') +
344 346                 Math.abs(count).toString();
345 347               };
347 348               Returns a serialized operation
348 349               as a string
349 350               */
350 351               @param {{op: string, start:
351 352               number, end: number, text:
352 353               string}}{op: string, start:
353 354               number, numChars: number}}
354 355               operation
355 356               An insertion or removal
356 357               object
357 358               @param {{op: string, start:
358 359               number, end: number, text:
359 360               string}}{op: string, start:
360 361               number, numChars: number}} |
361 362               prev]
362 363               The operation before this
363 364               operation
364 365               @returns {string} - The
365 366               serialized string for the
366 367               operation
367 368               @private
368 369               */
369 370               this._operationString = function (
370 371               operation, prev, base) {
371 372                 var offset, hack, operatorSnd;
372 373
373 374                 offset = Math.abs(count).toString();
374 375                 count = this.
375 376                 _lengthChangeString =
376 377                 function () {
377 378                   return (count > 0 ? '>' : '<') +
377 379                   Math.abs(count).toString();
378 380                 };
380 381                 Returns a serialized operation
381 382                 as a string
382 383                 */
383 384                 @param {{op: string, start:
384 385                 number, end: number, text:
385 386                 string}}{op: string, start:
386 387                 number, numChars: number}}
387 388                 operation
388 389                 An insertion or removal
389 390                 object
389 391                 @param {{op: string, start:
391 392                 number, end: number, text:
392 393                 string}}{op: string, start:
393 394                 number, numChars: number}} |
394 395                 prev]
395 396                 The operation before this
396 397                 operation
397 398                 @returns {string} - The
398 399                 serialized string for the
399 400                 operation
400 401                 @private
401 402                 */
402 403                 this._operationString = function (
403 404                 operation, prev, base) {
404 405                   var offset, hack, operatorSnd;
405 406
406 407                   offset = Math.abs(count).toString();
407 408                   count = this.
408 409                   _lengthChangeString =
409 410                   function () {
410 411                     return (count > 0 ? '>' : '<') +
410 412                     Math.abs(count).toString();
411 413                   };
413 414                   Returns a serialized operation
414 415                   as a string
415 416                   */
416 417                   @param {{op: string, start:
417 418                   number, end: number, text:
418 419                   string}}{op: string, start:
419 420                   number, numChars: number}}
420 421                   operation
421 422                   An insertion or removal
422 423                   object
423 424                   @param {{op: string, start:
424 425                   number, end: number, text:
425 426                   string}}{op: string, start:
426 427                   number, numChars: number}} |
427 428                   prev]
428 429                   The operation before this
429 430                   operation
430 431                   @returns {string} - The
431 432                   serialized string for the
432 433                   operation
433 434                   @private
434 435                   */
435 436                   this._operationString = function (
436 437                   operation, prev, base) {
437 438                     var offset, hack, operatorSnd;
438 439
439 440                     offset = Math.abs(count).toString();
440 441                     count = this.
441 442                     _lengthChangeString =
442 443                     function () {
443 444                       return (count > 0 ? '>' : '<') +
443 445                       Math.abs(count).toString();
444 446                     };
446 447                     Returns a serialized operation
447 448                     as a string
448 449                     */
449 450                     @param {{op: string, start:
450 451                     number, end: number, text:
451 452                     string}}{op: string, start:
452 453                     number, numChars: number}}
453 454                     operation
454 455                     An insertion or removal
455 456                     object
456 457                     @param {{op: string, start:
457 458                     number, end: number, text:
458 459                     string}}{op: string, start:
459 460                     number, numChars: number}} |
460 461                     prev]
461 462                     The operation before this
462 463                     operation
463 464                     @returns {string} - The
464 465                     serialized string for the
465 466                     operation
466 467                     @private
467 468                     */
468 469                     this._operationString = function (
469 470                     operation, prev, base) {
470 471                       var offset, hack, operatorSnd;
471 472
472 473                       offset = Math.abs(count).toString();
473 474                       count = this.
474 475                       _lengthChangeString =
475 476                       function () {
476 477                         return (count > 0 ? '>' : '<') +
476 478                         Math.abs(count).toString();
477 479                       };
479 480                       Returns a serialized operation
479 481                       as a string
481 482                       */
482 483                       @param {{op: string, start:
483 484                       number, end: number, text:
484 485                       string}}{op: string, start:
485 486                       number, numChars: number}}
486 487                       operation
487 488                       An insertion or removal
488 489                       object
489 490                       @param {{op: string, start:
490 491                       number, end: number, text:
491 492                       string}}{op: string, start:
492 493                       number, numChars: number}} |
493 494                       prev]
494 495                       The operation before this
495 496                       operation
496 497                       @returns {string} - The
497 498                       serialized string for the
498 499                       operation
499 500                       @private
500 501                       */
501 502                       this._operationString = function (
502 503                       operation, prev, base) {
503 504                         var offset, hack, operatorSnd;
504 505
505 506                         offset = Math.abs(count).toString();
506 507                         count = this.
507 508                         _lengthChangeString =
508 509                         function () {
509 510                           return (count > 0 ? '>' : '<') +
509 511                           Math.abs(count).toString();
510 512                         };
512 513                         Returns a serialized operation
513 514                         as a string
514 515                         */
515 516                         @param {{op: string, start:
516 517                         number, end: number, text:
517 518                         string}}{op: string, start:
518 519                         number, numChars: number}}
519 520                         operation
519 521                         An insertion or removal
521 522                         object
522 523                         @param {{op: string, start:
523 524                         number, end: number, text:
524 525                         string}}{op: string, start:
525 526                         number, numChars: number}} |
526 527                         prev]
527 528                         The operation before this
528 529                         operation
529 530                         @returns {string} - The
530 531                         serialized string for the
531 532                         operation
532 533                         @private
533 534                         */
534 535                         this._operationString = function (
535 536                         operation, prev, base) {
536 537                           var offset, hack, operatorSnd;
537 538
538 539                           offset = Math.abs(count).toString();
539 540                           count = this.
540 541                           _lengthChangeString =
541 542                           function () {
542 543                             return (count > 0 ? '>' : '<') +
542 544                             Math.abs(count).toString();
543 545                           };
545 546                           Returns a serialized operation
546 547                           as a string
547 548                           */
548 549                           @param {{op: string, start:
549 550                           number, end: number, text:
550 551                           string}}{op: string, start:
551 552                           number, numChars: number}}
552 553                           operation
553 554                           An insertion or removal
554 555                           object
555 556                           @param {{op: string, start:
556 557                           number, end: number, text:
557 558                           string}}{op: string, start:
558 559                           number, numChars: number}} |
559 560                           prev]
560 561                           The operation before this
561 562                           operation
562 563                           @returns {string} - The
563 564                           serialized string for the
564 565                           operation
565 566                           @private
566 567                           */
567 568                           this._operationString = function (
568 569                           operation, prev, base) {
569 570                             var offset, hack, operatorSnd;
570 571
571 572                             offset = Math.abs(count).toString();
572 573                             count = this.
573 574                             _lengthChangeString =
574 575                             function () {
575 576                               return (count > 0 ? '>' : '<') +
575 577                               Math.abs(count).toString();
576 578                             };
578 579                             Returns a serialized operation
579 580                             as a string
580 581                             */
581 582                             @param {{op: string, start:
582 583                             number, end: number, text:
583 584                             string}}{op: string, start:
584 585                             number, numChars: number}}
585 586                             operation
586 587                             An insertion or removal
587 588                             object
588 589                             @param {{op: string, start:
589 590                             number, end: number, text:
590 591                             string}}{op: string, start:
591 592                             number, numChars: number}} |
592 593                             prev]
593 594                             The operation before this
594 595                             operation
595 596                             @returns {string} - The
596 597                             serialized string for the
597 598                             operation
598 599                             @private
599 600                             */
600 601                             this._operationString = function (
601 602                             operation, prev, base) {
602 603                               var offset, hack, operatorSnd;
603 604
604 605                               offset = Math.abs(count).toString();
605 606                               count = this.
606 607                               _lengthChangeString =
607 608                               function () {
608 609                                 return (count > 0 ? '>' : '<') +
608 610                                 Math.abs(count).toString();
609 611                               };
611 612                               Returns a serialized operation
612 613                               as a string
613 614                               */
614 615                               @param {{op: string, start:
615 616                               number, end: number, text:
616 617                               string}}{op: string, start:
617 618                               number, numChars: number}}
618 619                               operation
619 620                               An insertion or removal
620 621                               object
621 622                               @param {{op: string, start:
622 623                               number, end: number, text:
623 624                               string}}{op: string, start:
624 625                               number, numChars: number}} |
625 626                               prev]
626 627                               The operation before this
627 628                               operation
628 629                               @returns {string} - The
629 630                               serialized string for the
630 631                               operation
631 632                               @private
632 633                               */
633 634                               this._operationString = function (
634 635                               operation, prev, base) {
635 636                                 var offset, hack, operatorSnd;
636 637
637 638                                 offset = Math.abs(count).toString();
638 639                                 count = this.
639 640                                 _lengthChangeString =
640 641                                 function () {
641 642                                   return (count > 0 ? '>' : '<') +
641 643                                   Math.abs(count).toString();
642 644                                 };
644 645                                 Returns a serialized operation
645 646                                 as a string
646 647                                 */
647 648                                 @param {{op: string, start:
648 649                                 number, end: number, text:
649 650                                 string}}{op: string, start:
650 651                                 number, numChars: number}}
651 652                                 operation
652 653                                 An insertion or removal
653 654                                 object
654 655                                 @param {{op: string, start:
655 656                                 number, end: number, text:
656 657                                 string}}{op: string, start:
657 658                                 number, numChars: number}} |
658 659                                 prev]
659 660                                 The operation before this
660 661                                 operation
661 662                                 @returns {string} - The
662 663                                 serialized string for the
663 664                                 operation
664 665                                 @private
665 666                                 */
666 667                                 this._operationString = function (
667 668                                 operation, prev, base) {
668 669                                   var offset, hack, operatorSnd;
669 670
670 671                                   offset = Math.abs(count).toString();
671 672                                   count = this.
672 673                                   _lengthChangeString =
673 674                                   function () {
674 675                                     return (count > 0 ? '>' : '<') +
674 676                                     Math.abs(count).toString();
675 677                                   };
677 678                                   Returns a serialized operation
678 679                                   as a string
679 680                                   */
680 681                                   @param {{op: string, start:
681 682                                   number, end: number, text:
682 683                                   string}}{op: string, start:
683 684                                   number, numChars: number}}
684 685                                   operation
685 686                                   An insertion or removal
686 687                                   object
687 688                                   @param {{op: string, start:
688 689                                   number, end: number, text:
689 690                                   string}}{op: string, start:
690 691                                   number, numChars: number}} |
691 692                                   prev]
692 693                                   The operation before this
693 694                                   operation
694 695                                   @returns {string} - The
695 696                                   serialized string for the
696 697                                   operation
697 698                                   @private
698 699                                   */
699 700                                   this._operationString = function (
700 701                                   operation, prev, base) {
701 702                                     var offset, hack, operatorSnd;
702 703
703 704                                     offset = Math.abs(count).toString();
704 705                                     count = this.
705 706                                     _lengthChangeString =
706 707                                     function () {
707 708                                       return (count > 0 ? '>' : '<') +
707 709                                       Math.abs(count).toString();
708 710                                     };
710 711                                     Returns a serialized operation
711 712                                     as a string
712 713                                     */
713 714                                     @param {{op: string, start:
714 715                                     number, end: number, text:
715 716                                     string}}{op: string, start:
716 717                                     number, numChars: number}}
717 718                                     operation
718 719                                     An insertion or removal
719 720                                     object
720 721                                     @param {{op: string, start:
721 722                                     number, end: number, text:
722 723                                     string}}{op: string, start:
723 724                                     number, numChars: number}} |
724 725                                     prev]
725 726                                     The operation before this
726 727                                     operation
727 728                                     @returns {string} - The
728 729                                     serialized string for the
729 730                                     operation
730 731                                     @private
731 732                                     */
732 733                                     this._operationString = function (
733 734                                     operation, prev, base) {
734 735                                       var offset, hack, operatorSnd;
735 736
736 737                                       offset = Math.abs(count).toString();
737 738                                       count = this.
738 739                                       _lengthChangeString =
739 740                                       function () {
740 741                                         return (count > 0 ? '>' : '<') +
740 742                                         Math.abs(count).toString();
741 743                                       };
743 744                                       Returns a serialized operation
744 745                                       as a string
745 746                                       */
746 747                                       @param {{op: string, start:
747 748                                       number, end: number, text:
748 749                                       string}}{op: string, start:
749 750                                       number, numChars: number}}
750 751                                       operation
751 752                                       An insertion or removal
752 753                                       object
753 754                                       @param {{op: string, start:
754 755                                       number, end: number, text:
755 756                                       string}}{op: string, start:
756 757                                       number, numChars: number}} |
757 758                                       prev]
758 759                                       The operation before this
759 760                                       operation
760 761                                       @returns {string} - The
761 762                                       serialized string for the
762 763                                       operation
763 764                                       @private
764 765                                       */
765 766                                       this._operationString = function (
766 767                                       operation, prev, base) {
767 768                                         var offset, hack, operatorSnd;
768 769
769 770                                         offset = Math.abs(count).toString();
770 771                                         count = this.
771 772                                         _lengthChangeString =
772 773                                         function () {
773 774                                           return (count > 0 ? '>' : '<') +
773 775                                           Math.abs(count).toString();
774 776                                         };
776 777                                         Returns a serialized operation
777 778                                         as a string
778 779                                         */
779 780                                         @param {{op: string, start:
780 781                                         number, end: number, text:
781 782                                         string}}{op: string, start:
782 783                                         number, numChars: number}}
783 784                                         operation
784 785                                         An insertion or removal
785 786                                         object
786 787                                         @param {{op: string, start:
787 788                                         number, end: number, text:
788 789                                         string}}{op: string, start:
789 790                                         number, numChars: number}} |
790 791                                         prev]
791 792                                         The operation before this
792 793                                         operation
793 794                                         @returns {string} - The
794 795                                         serialized string for the
795 796                                         operation
796 797                                         @private
797 798                                         */
798 799                                         this._operationString = function (
799 800                                         operation, prev, base) {
800 801                                           var offset, hack, operatorSnd;
801 802
802 803                                           offset = Math.abs(count).toString();
803 804                                           count = this.
804 805                                           _lengthChangeString =
805 806                                           function () {
806 807                                             return (count > 0 ? '>' : '<') +
806 808                                             Math.abs(count).toString();
807 809                                           };
809 810                                           Returns a serialized operation
809 811                                           as a string
811 812                                           */
812 813                                           @param {{op: string, start:
813 814                                           number, end: number, text:
814 815                                           string}}{op: string, start:
815 816                                           number, numChars: number}}
816 817                                           operation
817 818                                           An insertion or removal
818 819                                           object
819 820                                           @param {{op: string, start:
820 821                                           number, end: number, text:
821 822                                           string}}{op: string, start:
822 823                                           number, numChars: number}} |
823 824                                           prev]
824 825                                           The operation before this
825 826                                           operation
826 827                                           @returns {string} - The
827 828                                           serialized string for the
828 829                                           operation
829 830                                           @private
830 831                                           */
831 832                                           this._operationString = function (
832 833                                           operation, prev, base) {
833 834                                             var offset, hack, operatorSnd;
834 835
835 836                                             offset = Math.abs(count).toString();
836 837                                             count = this.
837 838                                             _lengthChangeString =
838 839                                             function () {
839 840                                               return (count > 0 ? '>' : '<') +
839 841                                               Math.abs(count).toString();
840 842                                             };
842 843                                             Returns a serialized operation
843 844                                             as a string
844 845                                             */
845 846                                             @param {{op: string, start:
846 847                                             number, end: number, text:
847 848                                             string}}{op: string, start:
848 849                                             number, numChars: number}}
849 850                                             operation
849 851                                             An insertion or removal
851 852                                             object
852 853                                             @param {{op: string, start:
853 854                                             number, end: number, text:
854 855                                             string}}{op: string, start:
855 856                                             number, numChars: number}} |
856 857                                             prev]
857 858                                             The operation before this
858 859                                             operation
859 860                                             @returns {string} - The
860 861                                             serialized string for the
861 862                                             operation
862 863                                             @private
863 864                                             */
864 865                                             this._operationString = function (
865 866                                             operation, prev, base) {
866 867                                               var offset, hack, operatorSnd;
867 868
868 869                                               offset = Math.abs(count).toString();
869 870                                               count = this.
870 871                                               _lengthChangeString =
871 872                                               function () {
872 873                                                 return (count > 0 ? '>' : '<') +
872 874                                                 Math.abs(count).toString();
873 875                                               };
875 876                                               Returns a serialized operation
876 877                                               as a string
877 878                                               */
878 879                                               @param {{op: string, start:
879 880                                               number, end: number, text:
880 881                                               string}}{op: string, start:
881 882                                               number, numChars: number}}
882 883                                               operation
883 884                                               An insertion or removal
884 885                                               object
885 886                                               @param {{op: string, start:
886 887                                               number, end: number, text:
887 888                                               string}}{op: string, start:
888 889                                               number, numChars: number}} |
889 890                                               prev]
8
```

```

87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173

offset = this._offsetString(
    operation, prev, base);
hack = operation.op == '+' ?
    '0' : '';

if (/^[\\n\\r]+$/.test(operation.
    text || '')) {
    operatorSnd = 'l+1'; // todo
    // Only works if charbank
    // == a single newline
} else {
    operatorSnd = operation.
        getOperation();
}

return offset + hack +
    operatorSnd;
};

/**
 * Returns the serialized charbank
 * string from all
 * operations
 * @returns {string}
 * @private
 */
this._charbankString = function ()
{
    var charbank, len, i;
    charbank = '';
    len = this._operations.length;
    for (i = 0; i < len; i += 1) {
        charbank += this.
            _operationCharbankString(
                this._operations[i]);
    }
    return charbank;
};

/**
 * Return the text of an operation
 * or an empty string

```

```

174 change += this._operations[i].
175   getLength() || 0;
176 }
177 if (change < 0) change += 3;
178 if (typeof base === 'string') {
179   return base.length + change;
180 }
181 if (base.textContent) {
182   return base.textContent.length
183     - 1 + change;
184 }
185 return null;
186 };
187 /**
188 * Returns if the sum of all
189   characters added and removed
190   in this
191   changeset
192   * @returns {number} - The sum of
193     all characters added and
194     removed
195     in this changeset
196     * @private
197     */
198 this._countLengthChange = function
199   () {
200     var change = 0, len, i;
201     len = this._operations.length;
202     for (i=0; i < len; i += 1) {
203       change += this._operations[i].
204         getLength() || 0;
205     }
206     return change;
207 };
208 /**
209 * Returns an array of all
210   operations of a changeset
211   ordered by the start offset
212   * @param {Type.Etherpad.Changeset}
213     changeset
214   * @returns {Array}
215   * @private
216   */
217 this._getOperations = function (
218   changeset) {
219   var operations = changeset.
220     getStack().slice(0);
221   operations.sort(this.
222     _compareOperations);
223   return operations;
224 };
225 /**
226 * Compares the offsets of two
227   insertions. This method can
228   be
229   * used with Array.prototype.sort
230   * @param {{start: number, end:
231     number, text: string}}{start
232     : number, numChars: number}}
233   a
234     An insertion or removal
235     object
236   b
237     An insertion or removal
238     object
239   * @returns {number}
240   * @private
241   */
242 this._compareOperations = function
243   (a, b) {
244     if (a.start < b.start) return
245       -1;
246     if (a.start > b.start) return
247       1;
248     return 0;
249   };
250 }
251 ).call(Type.Etherpad.
252   ChangesetSerializer.prototype);
253 module.exports = Type.Etherpad.
254   ChangesetSerializer;

```

Listing 43: /plugins/etherpad/changeset_serializer.js

Declaration of Academic Integrity

I hereby confirm that the present thesis on “A WYSIWYG Framework” is solely my own work and that if any text passages or diagrams from books, papers, the Web or other sources have been copied or in any other way used, all references – including those found in electronic media – have been acknowledged and fully cited.

.....

(Name, Date, Signature)