



university of  
 groningen

faculty of mathematics  
 and natural sciences

# In-line Editing: a New Approach to Editing Wikis

Bachelor's thesis

12th July 2011

Student: Jan Paul Posma

Supervisors: Leonie Bosveld-de Smet & Gerard R. Renardel de Lavalette

## Abstract

We present the concept of *in-line editing* pages in wikis; a compromise between editing raw wikitext and visually editing pages. This has the advantage of not requiring a formal grammar specification of the wikitext, a large problem for some wikis — including Wikipedia. Often, users of these wikis can only edit the raw wikitext, which makes editing a difficult task, especially for novice users who are used to visually editing documents.

With the proposed interfaces, users can click elements in the page, and then edit the wikitext for these elements, instead of having to edit the wikitext of the entire page at once, which can be frightening at first. We present three different interfaces based on in-line editing, and we look at usability experiments done with these interfaces.

We also present a way of generating these interfaces by leveraging the existing parser. We discuss an optimisation that allows for parsing only a part of the page when the user makes an edit, instead of having to parse the entire page again. We look at different ways of solving problems that arise when using our algorithm, such as the correct detection of editable elements, incorrect nesting that occurs after parsing, and resolving dependencies throughout the page. Finally, we consider ways of further improving this concept of in-line editing.

# Contents

<b>Contents</b>	<b>i</b>
<b>List of Figures</b>	<b>iii</b>
<b>List of Tables</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Related work</b>	<b>3</b>
2.1 Visual editor . . . . .	3
2.2 Traditional wikitext editor . . . . .	4
2.3 Visual editor with wikitext backend . . . . .	5
2.4 Wikitext editor with markup feedback . . . . .	6
2.5 Conclusions . . . . .	6
<b>3 New interfaces</b>	<b>8</b>
3.1 General concepts . . . . .	9
3.2 Functional editor . . . . .	9
3.3 Block editor . . . . .	10
3.4 Hybrid editor . . . . .	11
3.5 Conclusions . . . . .	12
<b>4 Usability experiments</b>	<b>13</b>
4.1 Testing the functional and the block editor . . . . .	13
4.1.1 Setup . . . . .	13
4.1.2 Analysis . . . . .	14
4.1.3 Results . . . . .	15
4.1.4 Conclusions . . . . .	16
4.2 Testing the hybrid editor . . . . .	17
4.2.1 Setup . . . . .	17
4.2.2 Analysis . . . . .	17
4.2.3 Results . . . . .	17
4.2.4 Conclusions . . . . .	18
<b>5 Algorithms</b>	<b>19</b>
5.1 Visual editing . . . . .	19
5.2 In-line editing . . . . .	20
5.3 Detecting elements . . . . .	21
5.4 Building a tree . . . . .	22
5.5 Marking wikitext . . . . .	24
5.6 Post processing . . . . .	24
5.7 Partial parsing . . . . .	26
5.8 Handling dependencies . . . . .	27
5.9 Browser . . . . .	28

5.10	Conclusions . . . . .	29
<b>6</b>	<b>Future work</b>	<b>30</b>
6.1	General interface enhancements . . . . .	30
6.2	Specialised interfaces . . . . .	30
6.3	Edit transactions . . . . .	31
6.4	Detection modules . . . . .	31
6.5	Robustness metrics . . . . .	32
6.6	In-line editing and visual editing . . . . .	32
6.7	Changing the wikitext syntax . . . . .	32
<b>7</b>	<b>Conclusions</b>	<b>33</b>
	<b>Bibliography</b>	<b>35</b>
<b>A</b>	<b>Usability testing</b>	<b>37</b>
A.1	Script . . . . .	37
A.1.1	Introduction and setup . . . . .	37
A.1.2	Explanation . . . . .	37
A.1.3	Tasks . . . . .	38
A.2	Form . . . . .	39
A.2.1	Questions . . . . .	39
A.2.2	Results . . . . .	39
A.3	Transcripts . . . . .	40
A.3.1	Participant 01 . . . . .	40
A.3.2	Participant 02 . . . . .	41
A.3.3	Participant 03 . . . . .	42
A.3.4	Participant 04 . . . . .	43
A.3.5	Participant 05 . . . . .	44
A.3.6	Participant 06 . . . . .	45
A.3.7	Participant 07 . . . . .	46
A.3.8	Participant 08 . . . . .	47
A.3.9	Participant 09 . . . . .	48
A.3.10	Participant 10 . . . . .	49
<b>B</b>	<b>Source code</b>	<b>50</b>
B.1	InlineEditorText.class.php . . . . .	50
B.2	InlineEditorPiece.class.php . . . . .	57
B.3	InlineEditorMarking.class.php . . . . .	58
B.4	InlineEditorRoot.class.php . . . . .	62
B.5	InlineEditorNode.class.php . . . . .	62
B.6	jquery.inlineEditor.js . . . . .	65
B.7	jquery.inlineEditor.editors.basic.js . . . . .	68
<b>C</b>	<b>Initial prototypes</b>	<b>72</b>
C.1	Prototype 1 . . . . .	72
C.2	Prototype 2 . . . . .	74
C.3	Prototype 3 . . . . .	77

# List of Figures

2.1	Visual editor with HTML backend . . . . .	4
2.2	Traditional wikitext editor . . . . .	4
2.3	Wikia visual editor . . . . .	6
2.4	WikiEd editor . . . . .	6
3.1	Sentence-level editing . . . . .	8
3.2	Yellow highlight . . . . .	9
3.3	Information box . . . . .	9
3.4	Functional editor edit modes . . . . .	10
3.5	Reference edit mode . . . . .	10
3.6	Block editor edit modes . . . . .	10
3.7	Paragraph edit mode in the block editor . . . . .	11
3.8	Hybrid editor . . . . .	12
4.1	Old editing text box . . . . .	14
4.2	Tasks for the first usability test . . . . .	15
5.1	Syntax tree . . . . .	20
5.2	Leveraging the existing parser . . . . .	20
5.3	Marking algorithm . . . . .	21
5.4	Markings in a paragraph . . . . .	22
5.5	Marking tree . . . . .	23
5.6	Partial parsing . . . . .	27
5.7	References in wikitext . . . . .	27
6.1	ProveIt tool . . . . .	31
C.1	Prototype 1 . . . . .	73
C.2	Prototype 2 . . . . .	74
C.3	Prototype 3 . . . . .	76

# List of Tables

4.1	Qualitative analysis . . . . .	15
A.1	Usability test 1 form results . . . . .	39

# Chapter 1

## Introduction

Wikis are one of the technological success stories of the last decade, and they are at the heart of the Web 2.0 revolution [15]. They allow any person within a company, in a community, or simply *anyone* at all, to change the content of webpages. The word ‘wiki’ is Hawaiian for ‘fast’, which suggests the ease of editing pages and collaborating online [24].

The most popular wiki is Wikipedia [1], but there are numerous public and private wikis that are used for a wide range of projects. They are used by corporations, governments and individuals, for internal organisation, collaboration within a community and with other parties, knowledge management, community websites, and much more [24].

Traditionally, wiki pages are stored as *wikitext*: plain text combined with some codes for markup. This markup was originally designed to make common tasks such as linking to other pages easy to do. This way, wikis can be used without much knowledge of technology. However, over the years wikitext implementations have become quite complicated.<sup>1</sup> Users have to deal with a variety of structures such as templates, references, and tables.

Wikipedia wants to increase both the quality of their content, and their number of contributors [22]. However, they have already hit a ceiling with their number of contributors in most languages in 2007 [16]. Since then, the total number of active<sup>2</sup> Wikipedia authors in all languages is quite stable [26]. Besides the difficulty of attracting new volunteers, another problem is that this current group of contributors is skewed towards young, male, and technologically skilled people [23].

A recent study shows that bad usability is a key factor in *wiki anxiety*, i.e. negative emotions that users might have when working with wikis [4]. This study, in which students used a wiki as part of a course, shows a significant negative correlation between wiki usability and wiki anxiety. Furthermore, there was no significant difference in wiki anxiety after the students used the wiki for several weeks. Other studies show that the complexity of wikitext gives rise to usability problems [6, 21].

To get a wider audience on board of Wikipedia, and other wiki projects, the usability of editing has to be improved to combat wiki anxiety [3, 4, 21, 22]. This can either be done by improving the editing interface, or by improving the wikitext syntax. While moving to an improved syntax certainly has its advantages and should be investigated, we look at making wikis more usable by rethinking the editing interface.

A common approach to solving the problem of building a usable editing interface, is by building a visual editor, like Microsoft Word or Google Docs. However, this can be very difficult if the parser — the software that converts wikitext to an actual webpage — does not use a mathematical description of the wikitext syntax, a so-called *formal grammar*. It turns out that it is indeed quite hard to build a visual editor for Wikipedia, as the current parser does not use such a formal grammar. There are

---

<sup>1</sup>For example, MediaWiki wikitext already mixes different kind of syntaxes, such as "traditional" wikitext and HTML: [http://meta.wikimedia.org/wiki/Help:Wikitext\\_examples](http://meta.wikimedia.org/wiki/Help:Wikitext_examples).

<sup>2</sup>In the statistics, contributors are called active when they make five or more edits per month.

many attempts to build visual editors, formal grammars, and alternative parsers, none of which are fully compatible with the current parser used on Wikipedia [11, 12, 14].

In this thesis, we present a compromise between visual editing and editing raw wikitext: *in-line editing*. It turns out that we can build in-line editing interfaces without making assumptions about the parser, which is a huge advantage for wikis for which it is hard to build a visual editor because of problems with the parser, such as not using formal grammar.

We first look at how a range of different existing editing interfaces apply to wikis (chapter 2). Then we present some new interfaces, based on the idea of in-line editing (chapter 3). We discuss usability tests done with these interfaces (chapter 4). Next, we consider the underlying algorithm that forms the basis of all these interfaces, and which can be applied to practically all existing or new wikis, independent of syntax (chapter 5). We look at the general approach of using an existing parser to generate these interfaces, and at the specific implementation and optimisations used in this research. We investigate the possibilities that this approach offers, and some challenges that need to be overcome (chapter 6). Finally, we present a summary of the findings, and some concluding remarks (chapter 7).



# Chapter 2

## Related work

In 2006, Christoph Sauer presented a new kind of editing interface for wikis [19]. In his paper, he compares editing raw wikitext with editing a visual representation, and shows how editing wikitext can be enhanced by adding visual elements to the text, similar to syntax highlighting. He also refers to an article by Leslie Lamport of 1986 [9]. Lamport argues against using anything other than a logical representation of the page, like wikitext. Sauer agrees with him, but he recognises the problems of using wikitext: it discourages new users, it allows for less control over layout, it causes a loss of overview, and the syntax can be quite complex.

In this chapter we look at the arguments of Sauer and Lamport, recent usability studies [6, 21], and existing implementations, to describe the pros and cons of different interfaces. Special attention is given to the implementation of these interfaces on existing wikis, as these have additional technological constraints, as we already saw in the previous chapter.

### 2.1 Visual editor

A visual editor<sup>1</sup> can be anything from Microsoft Word to Adobe Photoshop. With respect to wikis, one might think about a traditional visual editor to edit HTML, or even a live collaboration tool like Google Docs. There are a few severe drawbacks to this approach in most wiki contexts.

The most serious problem of using a fully visual editor is the loss of logical structure [9]. An extreme example would be a canvas where text and media elements could be placed and manipulated arbitrarily. The problem that immediately arises is how to deal with different screen resolutions, or even different devices. How would the page be presented to someone with a screen reader, an ancient mobile phone, or in printed form? Something that looks perfect on one computer screen might look quite different on other devices. With such a canvas there is no structure in the page, so elements cannot be arranged automatically in a way that suits a particular device.

A more sensible approach would be to apply some structure, as most online visual editors do. These editors typically use a formal language to save pages in. This has the disadvantage that power users who want to edit the source code have to do this in a language that is relatively hard to learn, such as HTML (Figure 2.1).<sup>2</sup>

---

<sup>1</sup>Often the term *WYSIWYG* — what you see is what you get — is used, but we will avoid this term. One reason is that in this context *WYSIWYG* refers to an editor with a wikitext source format, but we do not want to be limited to that. Another reason is that there are several similar abbreviations, which might be confusing. All in all, *visual editor* seems to be a more neutral expression.

<sup>2</sup>The Wikipedia article in the picture is *Tropical Depression Ten (2005)*, which we have used for many demonstrations, including the initial in-line editing prototypes. We chose this article because at the time it was a featured article, which means that it was of outstanding quality. In fact, it was the *smallest* featured article, which was a huge advantage, as we had to build the prototypes by hand.

**Tropical Depression Ten** was the tenth tropical cyclone of the record-breaking 2005 Atlantic hurricane season. It formed on August 13 from a tropical wave that emerged from the west coast of Africa on August 8. As a result of strong wind shear, the depression remained weak and did not strengthen beyond tropical depression status. The cyclone degenerated on August 14, although its remnants partially contributed to the formation of Tropical Depression Twelve, which eventually intensified into Hurricane Katrina.

(a) Rendered text.

```
<p>
<b>Tropical Depression Ten</b> was the tenth <a href="/wiki/Tropical_cyclone" title="Tropical cyclone">tropical cyclone
</a> of the record-breaking <a href="/wiki/2005_Atlantic_hurricane_season" title="2005 Atlantic hurricane season">2005
Atlantic hurricane season</a>. It formed on August 13 from a
<a href="/wiki/Tropical_wave" title="Tropical wave">tropical
wave</a> that emerged from the west coast of Africa on August
8. As a result of strong <a href="/wiki/Wind_shear" title=
"Wind shear">wind shear</a>, the depression remained weak and
did not strengthen beyond tropical depression status. The
cyclone degenerated on August 14, although its remnants
partially contributed to the formation of Tropical Depression
Twelve, which eventually intensified into <a href="/wiki/
Hurricane_Katrina" title="Hurricane Katrina">Hurricane
Katrina</a>.
</p>
```

(b) HTML source code (with syntax highlighting).

**Figure 2.1:** There are many existing editors that allow visual editing in the browser, which looks practically the same as the rendered text (a). But editing the source code requires learning HTML (b), which is relatively hard.

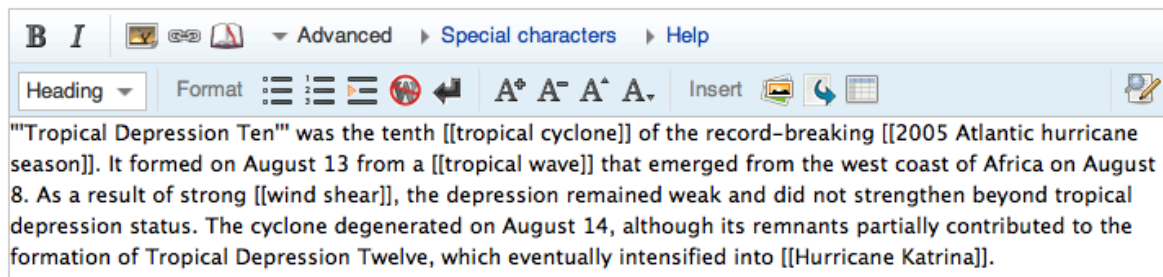
Another drawback is that with HTML — or another rich markup language — users can still make the page look exactly the way they want [9]. For them, this seems to be an advantage, as they can design pages to look exactly as they have in mind. But the consequence of this is that, again, it is hard to export to other output formats. To ensure transferability of data, users should be restricted to structuring the text logically, which limits their capability of visually formatting it. One can argue that this would make sense anyway, as most users have no expertise in designing layout.<sup>3</sup>

A visual editor also has some huge advantages. It allows users to directly interact with the document, and it is a method people are used to, because virtually all widely used office programs adopt this approach [19, 21].

## 2.2 Traditional wikitext editor

At the other end of the spectrum is the approach of only allowing users to edit the source code of pages. This is the kind of interface that is used by many large wikis, including Wikipedia. Instead of using a traditional markup language such as HTML as the storage and editing format, practically all wikis use wikitext, a special purpose language designed to be easy to learn and use. Although there are many variants, most wikitext syntaxes share some common characteristics. For example, instead of specifying a syntax for building tags, as with HTML (Figure 2.1b), special characters are defined for denoting

<sup>3</sup>To keep in line with the wiki spirit, it should be possible to do layout designing for users that happen to have this expertise. This should not be done in the normal pages or articles, but for example in style files and templates.



**Figure 2.2:** Traditional wikitext editor, with toolbar.

emphasis, links, headings, and so on (Figure 2.2). In this thesis, we will talk about wikitext as if only one syntax exists, even though there are different dialects.

Wikitext is typically much more restricted than traditional markup languages, so users can only specify the content of the page, not what it will look like.<sup>4</sup> By only allowing a limited set of codes like in Figure 2.2, the source code of the page also becomes more readable. In fact, wikitext syntax is designed to be as intuitive and non-intrusive as possible.

There are many advantages to editing pure wikitext, such as the focus on content instead of form, the speed with which users can edit pages, and the ease of displaying the page in different output formats [19]. Wikitext syntax is designed to be easier to learn than traditional markup languages: a study has shown that even 8-year-olds learn wikitext syntax in a very short amount of time [6].

There are also problems with editing wikitext. The same study with the 8-year-olds shows that there is a significant number of problems that have to do with the wikitext syntax [6], a finding supported by recent studies by the Wikimedia Foundation [21]. Novice users are often intimidated by the syntax. They might lose interest quickly, especially when they contribute voluntarily. If the interface looks too hard at first sight, there is no incentive to continue editing.

Another problem is that it is easy to lose the overview [19, 21]. Even seasoned contributors experience problems to find a single line they want to edit. With the large amounts of text involved, it can be hard to see the structure of the page. MediaWiki, the software behind Wikipedia, solves this partly by allowing users to edit only one section, but this is no perfect solution, as a single section can still contain a lot of text and codes.

The problem that the user initially does not know the syntax, can be overcome by the use of a toolbar that can apply the basic markup features to the text [19, 21]. Help pages — perhaps even integrated with this toolbar — can also assist the user in learning the syntax.

On the other hand, research shows that when users want to learn the syntax, they tend to look around on pages to find similar markup, and then copy the syntax used there [21]. Also, to check whether the syntax is used correctly, users have to switch back and forth between the editing interface showing the source, and a preview page or tab, which can be confusing, especially for novice users [21].

## 2.3 Visual editor with wikitext backend

Building a visual editor on top of wikitext allows both novice users and power users to work with an interface that suits them well, which is the reason that a lot of wikis already use such an interface.<sup>5</sup> For example, the site Wikia that hosts many small wikis provides such an interface, as shown by Figure 2.3. By using a visual editor, novice users will not be discouraged at first because they do not see the underlying format, but an interface they are used to when editing text in general. Power users are still able to edit raw wikitext to make more complex edits, for example in templates or references.

While this is a good solution for new wikis, it is harder to implement such an interface for some existing wikis. There are wikis — including Wikipedia — for which there is no formal grammar describing the wikitext language, thus making it difficult to create a client implementation of such a parser that runs in the browser in real time [14].

Power users can be very creative when it comes to stretching the possibilities of a specific parser implementation to the limit.<sup>6</sup> Porting an existing *de facto* parser to client side code, or even trying to formalise the existing parser in a grammar definition, are both cumbersome and error-prone tasks, as developers have to take all these edge cases into account, even though they might not be documented at all. Even worse is that when a page is parsed incorrectly, this could make the entire page un-editable,

---

<sup>4</sup>Many implementations have codes for bolds and italics, but these can also be regarded as different levels of emphasis.

<sup>5</sup>Most wiki farms on this list use this type of editing interface:  
[http://en.wikipedia.org/wiki/Comparison\\_of\\_wiki\\_farms](http://en.wikipedia.org/wiki/Comparison_of_wiki_farms).

<sup>6</sup>One particularly interesting example is the implementation of a widely used string length function on Wikipedia, expressed in parser functions that allow padding of text with whitespace:  
[http://en.wikipedia.org/wiki/Template:Str\\_len/core](http://en.wikipedia.org/wiki/Template:Str_len/core).



Tropical Depression Ten was the tenth tropical cyclone of the record-breaking 2005 Atlantic hurricane season. It formed on August 13 from a tropical wave that emerged from the west coast of Africa on August 8. As a result of strong wind shear, the depression remained weak and did not strengthen beyond tropical depression status. The cyclone degenerated on August 14, although its remnants partially contributed to the formation of Tropical Depression Twelve, which eventually intensified into Hurricane Katrina.

(a) Visual editor.



```
'''Tropical Depression Ten''' was the tenth [[tropical cyclone]] of the record-breaking [[2005 Atlantic hurricane season]]. It formed on August 13 from a [[tropical wave]] that emerged from the west coast of Africa on August 8. As a result of strong [[wind shear]], the depression remained weak and did not strengthen beyond tropical depression status. The cyclone degenerated on August 14, although its remnants partially contributed to the formation of Tropical Depression Twelve, which eventually intensified into [[Hurricane Katrina]].
```

(b) Wikitext source code.

**Figure 2.3:** The editor used by Wikia, which allows visual editing like users expect (a), but also enables power users to edit the page using wikitext (b).

which is highly undesired. The lack of a formal grammar is so much of a problem for some wikis, that the Wikimedia Foundation recently decided to write one for MediaWiki after all, in a large multi-year project [12].

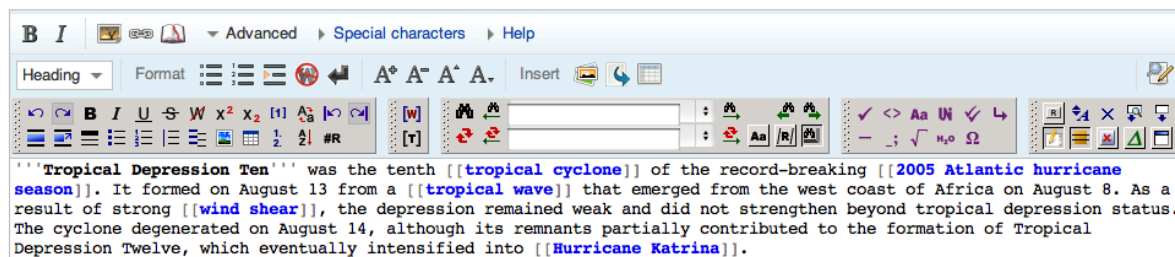
## 2.4 Wikitext editor with markup feedback

One way to deal with the issue of wikitext complexity when editing raw wikitext instead of a visual representation, is to apply a technique well known to computer scientists: syntax highlighting. For decades this has been an excellent way to edit program code, and it can be applied to wikitext as well. To take it even further, we can make the visual feedback mimic the rendered page (Figure 2.4), so that the user can better understand the meaning of different codes [19].

This still does not completely solve the problem for existing wikis, as a subset of the parsing grammar still has to be implemented. The advantage of this technique, however, is that this implementation does not have to be perfect. When wikitext is parsed incorrectly, the worst thing that could happen is that the highlighting will be incorrect, whereas with a fully visual editor the entire page could become un-editable.

## 2.5 Conclusions

While wikitext is designed to be intuitive and easy to learn, it still causes usability problems. It is not without reason that many sites, like Wikia, use a visual editor with wikitext backend. This allows novice



**Figure 2.4:** WikiEd, an editor featuring syntax highlighting and mimicking of the rendered page.

users to work with the visual representation, like they are used to do when editing text documents. On the other hand, power users are allowed to use the wikitext representation, which still is relatively easy to learn and use compared to traditional markup languages such as HTML. When editing the raw wikitext, it is also possible to aid the user by highlighting the text in a way similar to syntax highlighting of source code, and by even mimicking how elements will look when the page is rendered. However, power users still face the problem of losing the overview.

Moreover, for some wikis it is barely possible to deploy a visual editor with wikitext backend, because of the lack of a formal grammar specification. Such wikis are stuck with a traditional wikitext editor, at the most with some syntax highlighting and style mimicking.

This research emerged from the need for an interface that allows direct manipulation of the page, without depending on a formal grammar. The concept of in-line editing that we present is a compromise between visual editing, and editing wikitext. Incidentally, it also solves the problem of losing the overview for power users of wikis that do actually use a visual editor, as we see in the next chapter.

## Chapter 3

# New interfaces

In this paper, we propose interfaces in which the entire page is shown to the user as if it is rendered for viewing, but with elements in the page highlighted so that they can be edited. When an element is clicked, a popup is shown, containing a text box (Figure 3.1). The user will still edit plain wikitext, but does not have to see all the wikitext of the page at once, which should be less intimidating. Power users will benefit as well, because they do not have to search around in the wikitext: as they can navigate through the original page, there is no problem of losing the overview. We call this approach of editing wikitext inside the original page *in-line editing*.

By allowing users to preview directly in the page, such an editing interface feels more like a visual editor, which most people are used to. It allows for direct manipulation of the page, without having to switch between a preview window and the editor. Finally, this interface can be implemented without having a formal grammar of the wikitext syntax, which is a major problem that prevents some wikis from using a visual editor. We see how this works precisely in chapter 5.

We have created three different interfaces all based on the idea of in-line editing. In this chapter we first look at the general concepts all these interfaces share. We then discuss the three different interfaces in the order they were conceived. The first interface is the *functional editor*, which we showed to the MediaWiki mailing list even before the back-end was implemented.<sup>1</sup> We posted it just to see how the community would respond, as this is vital in open-source development. After a positive response, we implemented it as an extension in MediaWiki. Then the idea of the *block editor* was formed, which is similar to the functional editor but takes a different approach. After we implemented this interface as well, we did a usability experiment with both of these interfaces, as we see in the next chapter. Based on the results of this experiment, we decided to merge these two interfaces, creating the *hybrid editor*.

<sup>1</sup>For more information about the first prototypes, see Appendix C.

### Tropical Depression Ten (2005)

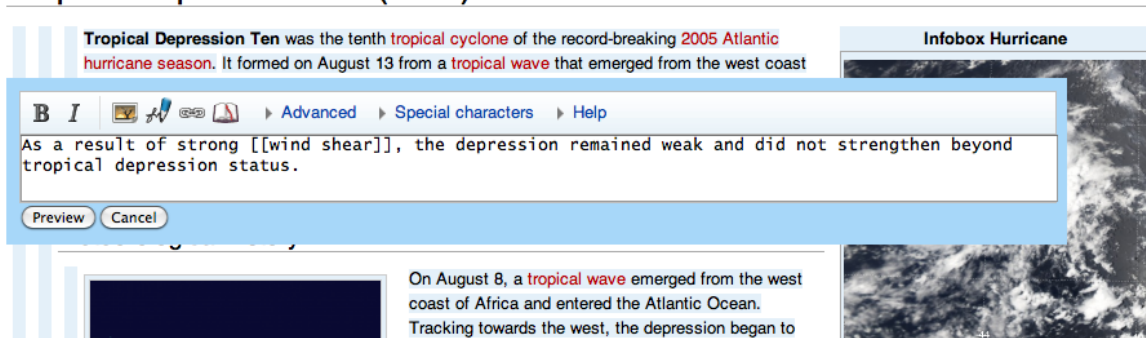


Figure 3.1: Basic sentence-level editing approach. A popup allows for editing of one sentence.

## 3.1 General concepts

All interfaces share the basic idea of allowing the user to click on elements in the page in order to edit the wikitext of these elements. Elements that are clickable and thus editable have a light blue background colour, which changes to a slightly darker colour when the mouse pointer hovers over them. When an element is clicked, a popup window is shown, which contains a text box, along with preview and cancel buttons. After editing the wikitext, the user can click the preview button, and the page is updated to reflect the changes. The changed element gets a yellow highlight to indicate the change (Figure 3.2).

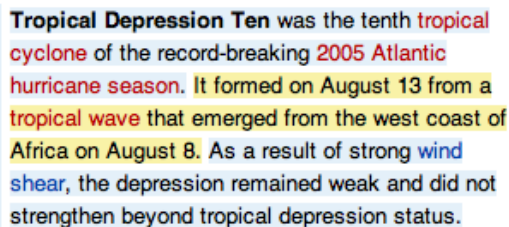
This in-line approach of editing content on the web is not new, and has been described by Michael Rees in 2000 [18]. In his paper, he describes various techniques of implementing in-line editing, and concludes that this user interface is surprisingly powerful and effective. However, the interfaces that he mentions allow editing of fixed elements in the page, instead of elements in dynamic content as we see in wikis. Enabling users to edit a wiki page by editing the wikitext belonging to dynamic elements in the page, is a new approach.

The editing of individual sentences, so-called *sentence-level editing*, is also new. The main reason to do this is to avoid that users have to deal with the syntax of references, as they would when editing entire paragraphs. References are placed between sentences, and often contain templates, which could be intimidating for novice users. Editing of individual sentences, without the references, is a neat way of avoiding this problem.

After making a number of changes, the user can decide to save the page. On the top of the page a blue information box is shown, containing a button to publish the page (Figure 3.3). This box also contains the edit summary box, where the user should describe the changes. The information box also contains some basic instructions and a legal disclaimer, but besides that it is deliberately kept as simple as possible.

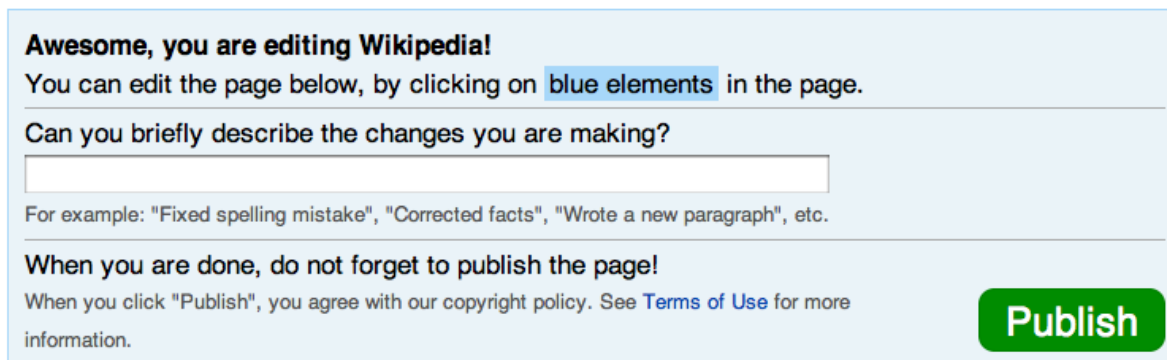
## 3.2 Functional editor

With the functional editor, the user can select different edit modes by using buttons at the top of the page. By default the option *text* is selected, but the user can proceed to select *lists*, *references*, *media*, *templates*, or *full editor* (Figure 3.4). The edit mode that is selected defines which elements in the page are actually editable. In the default mode, *text*, only individual sentences are highlighted and thus



Tropical Depression Ten was the tenth tropical cyclone of the record-breaking 2005 Atlantic hurricane season. It formed on August 13 from a tropical wave that emerged from the west coast of Africa on August 8. As a result of strong wind shear, the depression remained weak and did not strengthen beyond tropical depression status.

**Figure 3.2:** A yellow highlight is a nice way of showing edits.



**Awesome, you are editing Wikipedia!**  
You can edit the page below, by clicking on blue elements in the page.

Can you briefly describe the changes you are making?

For example: "Fixed spelling mistake", "Corrected facts", "Wrote a new paragraph", etc.

**When you are done, do not forget to publish the page!**  
When you click "Publish", you agree with our copyright policy. See [Terms of Use](#) for more information.

**Publish**

**Figure 3.3:** Information box, at the top of each editing interface.



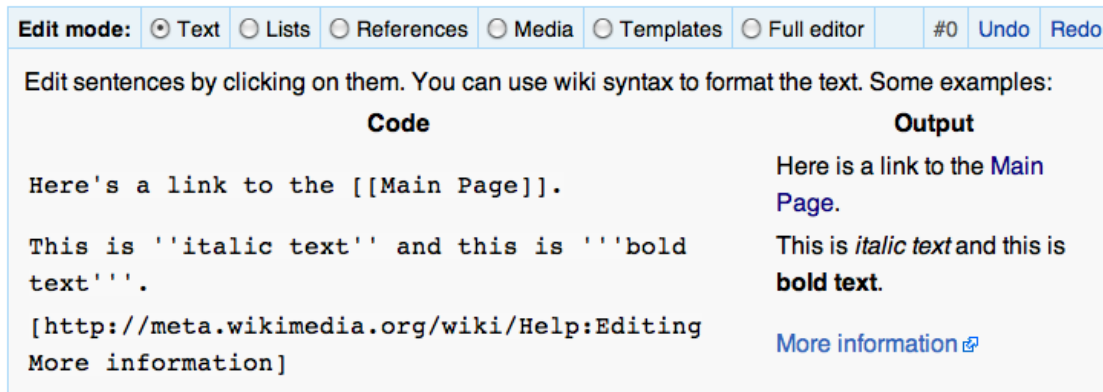


Figure 3.4: Edit modes in the functional editor, with the *text* mode selected.

editable. When a different mode is selected, different parts of the page light up with the blue background colour.

The rationale for this type of interface is that the user can learn more about wikitext, by explicitly selecting a particular edit mode. This way the user can start with simple constructs such as sentences and lists, and advance to complex syntax such as references and templates (Figure 3.5). It is also possible to give the user guidance, by adapting the editing process to the specific edit modes. For example, in our implementation we show a short explanatory text for each mode, with links to pages with more information.

The downside of this approach is that, while it should be clear how to edit something like a reference or template, it is not obvious how to add one. This could be done by switching to any other mode, and then appending the required wikitext at the end of an editable marking. As the user is never presented with more than a few lines of wikitext, especially novice users will not know the structure of a page. It might be hard for them to figure out that adding wikitext by appending to another element is even possible.

### 3.3 Block editor

With the block editor we take yet a different approach. We split up the page in elements of different sizes. The user can, again, select an edit mode. This time, the user can switch between *text*, *paragraphs*, *sections*, and *full editor* (Figure 3.6). This solves that users only see a couple of lines when editing the

gns of **convective organization** on August 11. The system continued to c  
 and at 1200 UTC on August 13. <sup>[2]</sup> Upon its designation, the depression  
 ved **banding features** and expanding **outflow**. <sup>[3]</sup> The depression move  
 f any significant intensification.

Figure 3.5: In the reference edit mode only references are editable.

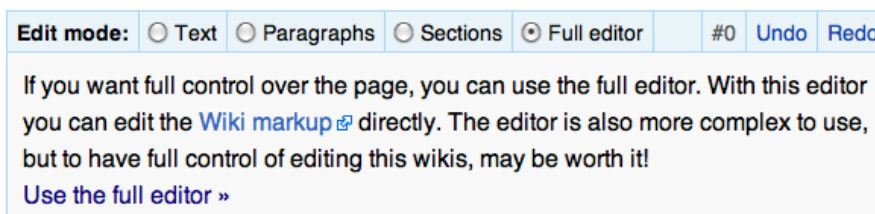
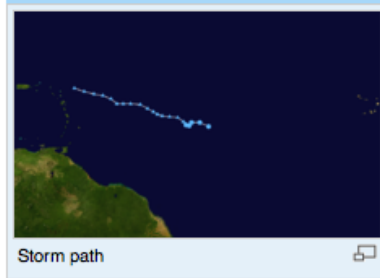


Figure 3.6: Edit modes in the block editor, with the *full editor* mode selected. This mode allows to use the original editor, the traditional wikitext editor.



## Meteorological history



On August 8, a **tropical wave** emerged from the west coast of **Africa** and entered the **Atlantic Ocean**. Tracking towards the west, the depression began to exhibit signs of **convective organization** on August 11. The system continued to develop, and it is estimated that Tropical Depression Ten formed at 1200 UTC on August 13. <sup>[2]</sup> Upon its designation, the depression consisted of a large area of thunderstorm activity, with curved **banding features** and expanding **outflow**. <sup>[3]</sup> The depression moved erratically and slowly towards the west, and **wind shear** inhibited any significant intensification.

<b>Fatalities</b>	None reported
<b>Damage</b>	Unknown
<b>Areas affected</b>	None
Part of the <b>2005 Atlantic hurricane season</b>	

By early August 14, the shear had substantially disrupted the storm, leaving the low-level center of circulation exposed from the area of convection, which was also deteriorating. After meandering, the storm began to move westward. <sup>[4]</sup> By 1800 UTC on August 14, the strong shear had further weakened the storm, and it no longer met the criteria for a tropical cyclone. It degenerated into a remnant low, and the National Hurricane Center issued their final advisory on the cyclone. <sup>[2]</sup>

<sup>[5]</sup> <sup>[6]</sup> In a re-analysis, it was found that the low-level circulation of Tropical Depression Ten had completely detached and dissipated; only the remnant mid-level circulation moved on and merged with the second tropical wave. As a result, the criteria for keeping the same name and identity were not met. <sup>[7]</sup>

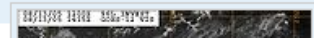


Figure 3.7: Paragraph edit mode in the block editor

page and thus do not see how the page is structured, as we see with the functional editor. By enabling users to gradually work with larger chunks of wikitext, they can discover the structure of the page (Figure 3.7).

With this editor however, it is harder to give specific guidance for the different elements in the wikitext. There still is a learning element, as the user can progress from small portions of wikitext — containing less codes — to large sections or even the entire page at once.

Note that in both the functional and the block editor, the edit mode *full editor* (Figure 3.6) takes the user back to the old, traditional editing interface as seen in section 2.2. This was done to make sure the page would still be editable in case one of these interfaces failed for some reason. The idea was that this fallback mechanism could be removed when the interface proved to be stable, and thus it was eventually removed in the hybrid editor as we see in the next section.

## 3.4 Hybrid editor

In the next chapter we look at a usability experiment done with both the functional editor and the block editor. One of the conclusions of that experiment is that the edit modes do not work as intended. With this conclusion, a logical next step is to remove them. For the functional editor this would mean simply enabling the editing of all the edit modes at once. There is no problem of overlap, as all the editable elements are disjoint.<sup>2</sup>

For the block editor, overlap is a problem when trashing the edit modes. We cannot enable editing of all elements without changing the interface. Because when clicking on a sentence, would we then edit the sentence, the entire paragraph, or the section? A way to solve this is to present a tree-like structure at the left side of the page. This is implemented by not highlighting entire paragraphs and sections, but only showing a line marking on the left side of each element.


<sup>2</sup>Elements are disjoint, but with some minor exceptions, such as templates inside templates, templates inside sentences, and references inside templates, and so on. When this happens, we just remove the innermost elements, as discussed in section 5.4.

## Tropical Depression Ten (2005)

**Tropical Depression Ten** was the tenth **tropical cyclone** of the record-breaking **2005 Atlantic hurricane season**. It formed on August 13 from a **tropical wave** that emerged from the west coast of Africa on August 8. As a result of strong **wind shear**, the depression remained weak and did not strengthen beyond tropical depression status.

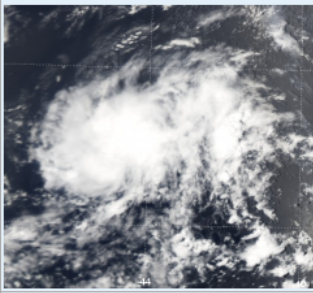
The cyclone degenerated on August 14, although its remnants partially contributed to the formation of Tropical Depression Twelve, which eventually intensified into **Hurricane Katrina**.<sup>[1]</sup> The cyclone had no effect on land, and did not directly result in any fatalities or damage.

### Meteorological history



On August 8, a **tropical wave** emerged from the west coast of Africa and entered the Atlantic Ocean. Tracking towards the west, the depression began to exhibit signs of **convective organization** on August 11. The system continued to develop, and it is estimated that Tropical Depression Ten formed at 1200 UTC on August 13. At the time, it was located about **Template:Convert** east of **Barbados**.<sup>[2]</sup> Upon its designation, the depression consisted of a large area of thunderstorm activity, with

### Infobox Hurricane



<b>Formed</b>	August 13, 2005
<b>Dissipated</b>	August 14, 2005
<b>Highest winds</b>	35 mph (55 km/h)
<b>Lowest pressure</b>	Unknown

**Figure 3.8:** Hybrid editor, with a paragraph selected. The leftmost line marking highlights the entire page, the middle one highlights the section, and next to that are the paragraph markings.

As the interface of the block editor is moved to the side of the page, it makes sense to combine it with the new functional editor, which still operates inside the page. This new combined interface is the hybrid editor (Figure 3.8).

When implementing this interface in MediaWiki, it became clear that this was a more sensible solution all along. The code became a lot simpler and better organised, and hundreds of lines of interface-specific code could be removed.

With this interface, there is no need for a button for the full editor anymore, as this can simply be implemented as another line marking on the left spanning the entire article. The interface also features even fewer instructions at the top of the page, while keeping the advantage of both editors. The user can still understand the structure of the page by clicking lines on the left, and there can also be specific guidance for elements like references, images and templates when clicking such an element inside the article.

## 3.5 Conclusions

With in-line editing, we introduce an existing technique to dynamic pages like wikis. Sentence-level editing is useful for editing even smaller pieces of wikitext, and for avoiding confronting users with references. Different interfaces can be built with the in-line editing concept, and the hybrid editor is a nice example of an interface that evolved from other interfaces. Still, there might be an even better solution that we have not yet discovered.

In the next chapter we see how we actually came to the conclusion that edit modes do not work, and which led to the idea of the hybrid editor. We also discuss a usability experiment done with the hybrid editor.

# Chapter 4

## Usability experiments

In this chapter we first look at a usability experiment we did with the functional editor and the block editor. This experiment eventually led to the development of the hybrid editor. We then discuss a usability experiment done by other researchers, who tested the hybrid editor.

### 4.1 Testing the functional and the block editor

In November 2010 we performed a usability experiment with the functional editor and the block editor. We also had a control group of users who used the traditional wikitext editor of MediaWiki, which we call the *original editor* for short. The ten students who took part in the experiment were asked to perform a few editing tasks on a wiki that resembled Wikipedia. They had 15 minutes to complete these tasks. The experiment took place at the University of Groningen, with both master's and bachelor's students of Information Science.<sup>1</sup>

#### 4.1.1 Setup

Before the start of the experiment, all participants signed a declaration of informed consent. In this consent they also agreed to anonymous publication of an audio and video recording of the experiment online. They received basic instructions on how the experiment worked, what was expected of them, and about the goals. They were told that they could do nothing wrong, and that the interfaces were being tested, not the participants themselves.

For the experiment a Dutch instance of MediaWiki was set up, with a shortened copy of the Wikipedia lemma about the city of Groningen.<sup>2</sup> Users were asked to perform tasks of increasing difficulty, while thinking aloud.<sup>3</sup> The participants first had to fix a typographical error and add a sentence. Then they were asked to add an external link, add a reference, and eventually to edit a template call. The actual tasks have been adapted from the moderation script used by the Wikimedia Foundation for their usability research [21]. More details are available in Appendix A.

Of the five master's students, one was asked to use the original editor, two used the functional editor, and another two used the block editor. The same distribution was supposed to be used for the bachelor's students, but because of a technical problem the last user used the block editor instead of the functional editor.

The main objective was to see whether the editing of wiki pages became any easier with one of the new interfaces, and how this would differ between the tasks. Of the ten participants, only two had previously

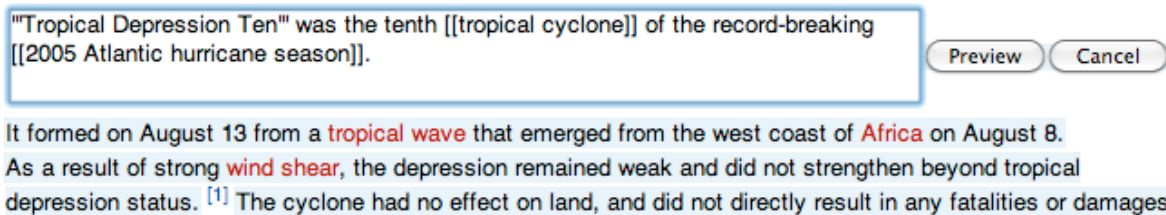
---

<sup>1</sup>Some information about these courses:

<http://www.rug.nl/let/onderwijs/afdelingen/informatiekunde/index?lang=en>.

<sup>2</sup>Original article: [http://nl.wikipedia.org/wiki/Groningen\\_\(stad\)](http://nl.wikipedia.org/wiki/Groningen_(stad)).

<sup>3</sup>We used the standard think-aloud protocol: [http://en.wikipedia.org/wiki/Think\\_aloud\\_protocol](http://en.wikipedia.org/wiki/Think_aloud_protocol).



**Figure 4.1:** A popup window was not yet implemented in these interfaces. Instead, users edited wikitext using a text box that appeared between the text.

clicked the edit button in Wikipedia, but had not attempted to make any changes. Thus, they would all learn a bit about wikitext during the 15 minutes that the experiment lasted. The secondary goal was to see whether or not the participants would use the different edit modes during their learning process as we expected.

Note that in this context participants were given specific tasks, which may result in a different learning process than when users would explore without set goals. Thinking aloud may also influence the way one acts [8]. Besides that, the group of subjects was nowhere near representative of all internet users, being young, intelligent, and having experience with programming. Therefore, only shallow conclusions can be drawn, but this is enough to discover the major flaws in the interfaces.

At the end of the experiment, all participants were asked to fill in a short survey which asked for their personal details, as well as their Wikipedia usage and suggestions for the editing interface they used.

Note that in the version we tested, the popup window and editing toolbar as shown in Figure 3.1 on page 8 were not used. Users had to edit the wikitext using a text box that appeared between the text, instead of in a popup window (Figure 4.1).

## 4.1.2 Analysis

Because of the small and biased groups, we do not look at any quantitative measurements. Instead, we look at general trends that were noticeable among the different groups. In Table 4.1 the tasks are rated with respect to the ease of their performance by each user. Besides the task ratings, the most occurring problems are listed per task. For easy reference, a summary of the tasks is included in Figure 4.2.

For the task ratings we have only looked at how difficult the task was once the user got to the editing interface. It happened sometimes that the user had problems reaching the edit page altogether. We have not included this in the task rating, but we have marked it in the table.

There are quite a few tasks where the user used the *full editor* edit mode. This mode takes the user to the original editor, where the user can edit the page without using in-line editing, as noted in section 3.3. When looking at the results it is necessary to differentiate between tasks performed with this interface, and those performed with an in-line editing interface. Therefore, we have marked it in the table when users used this edit mode.

Because of the limited time for each test, it was sometimes necessary to give a hint, which we have marked in the table as well. If users would have done these tasks on their own, some would probably have figured out how the interface works anyway, while others might have quit at that point.

When the user says he or she would have given up, or actually gives up during the test, we have also marked this. We usually continued the test anyway, often after giving a hint.

**Table 4.1:** Qualitative analysis

Editor Participant	Original		Functional			Block				
	01	08	03	05	06	02	04	07	09	10
Typo	*	++ <sup>g</sup>	++	+ <sup>cfh</sup>	++ <sup>fg</sup>	++ <sup>f</sup>	++ <sup>fg</sup>	++ <sup>g</sup>	++ <sup>g</sup>	++
Sentence	- <sup>gh</sup>	++	++	++	++	++	- <sup>c</sup>	++	++	++
Link	++	++	- <sup>bde</sup>	*	- <sup>abc</sup>	- <sup>a</sup>	*	+ <sup>a</sup>	++ <sup>d</sup>	++
Reference	- <sup>bd</sup>	+ <sup>d</sup>	-- <sup>ce</sup>	*	+ <sup>a</sup>	-- <sup>cd</sup>	- <sup>be</sup>	+ <sup>ad</sup>	- <sup>abd</sup>	- <sup>bd</sup>
Template	+ <sup>bci</sup>	++	+ <sup>e</sup>	-- <sup>bi</sup>	+ <sup>a</sup>	+ <sup>ah</sup>	++	++	+ <sup>a</sup>	++

++: easy; +: doable; -: hard, doable with hints; --: not doable.

\* Task was not done.

<sup>a</sup> User uses the full editor (traditional wikitext editing).

<sup>b</sup> Giving a hint was required.

<sup>c</sup> Gives up or would have given up.

<sup>d</sup> Problems with the wikitext syntax.

<sup>e</sup> User is confused by the different edit modes.

<sup>f</sup> Scrolls to the bottom looking for publish button.

<sup>g</sup> User does not enter an edit summary.

<sup>h</sup> Problems with previewing of the page.

<sup>i</sup> Already had problems before going to the editing interface.

### 4.1.3 Results

**Typo** The first task the users had to complete was to fix a typographical error. All users first had to locate the correct edit button. Most users saw the edit buttons at the side of each section, but the mistake they had to fix was in the introduction, which lacked such a button. Some users randomly clicked on the wrong edit button, but most found the tab at the top eventually.

The control group found the typo with relative ease. They were not immediately put off by the large amount of wikitext, which might be due to their technical background. Another important issue is that none of the users in the control group, and a minority of the users who used the full editor later on, read any of the instructions on the page. Most users left the edit summary text box empty.

In contrast, most of the users of both the functional and block editor read the instructions before starting to edit. They would then find the edit summary text box which has better visibility in these interfaces. The editing of a sentence was really easy for all users. Some were a bit surprised to see codes when clicking a sentence, but this was not a problem for anyone.

**Sentence** Adding a sentence was an easy task for almost all users. One user struggled with the original editor, and clicked the wrong edit link. Only after finding out that the wrong section was being edited, this user quickly edited the correct section. One block editor user struggled with putting the sentence after a reference instead of before one.

**Link** The real difficulties began when adding an external link. Users of the original editor had no difficulty with locating the correct section, and then copying the syntax. Of the two functional editor users who got this task, both were confused as to which edit mode to select, one selected the correct one, and one deferred to the full editor. Of the block editor users, only two used the edit modes correctly; both of them used section editing. The other users did not notice the edit modes, or were not willing to click them to try them out, and ended up using the full editor.

**Typo** Fix a typographical error.

**Sentence** Add a sentence to a paragraph.

**Link** Add an external link.

**Reference** Add a reference (citation) in the text.

**Template** Increase the population count in an info box.

**Figure 4.2:** User tasks

Some users that used the full editor had trouble previewing the page. They clicked the preview tab correctly, but were confused as they had to locate the part of the page they edited. Getting back to the editing tab was also confusing, as some users forgot they were looking at the preview tab, and tried to go back by clicking the back button of the browser. With the functional and block editors, these problems did not occur.

**Reference** The most difficult task was adding a reference. Only a few participants figured this out on their own, some failed completely, and most of them needed help. Many users used the full editor eventually, although three block editor users managed to do the task without it. One of them found the syntax online and used it, another one used the paragraph edit mode, and yet another used the section edit mode. One user even mentioned that perhaps the way references are organised is similar to how they are organised in LaTeX, which is indeed the case. But still, he failed to add a reference, without trying to look for the codes in the text.

Even the users who used the functional editor failed to use the reference edit mode correctly. None of them noticed that references were highlighted in the text (Figure 3.5 on page 10), and they did not understand the instructions that were given to them. Some users mistook the category codes for elements of the reference list.<sup>4</sup>

**Template** The last task was to change the population count of Groningen both in the text and in a template call, an info box. Changing the count in the text was not a problem for anyone. When changing it in the info box, a common mistake was that users clicked a link inside the info box, which would take them to a page about populations. All users were able to change the number eventually, either by using the full editor, or the appropriate edit mode.

**Form after the test** Users were asked to fill in a form after the test, in which they could fill in their personal details, and suggestions to the interface. None of the suggestions were new ideas, as they included visual editing, syntax highlighting, making the editing of references easier, less instructions to read, and so on. The full results of this form are available in Appendix A.

#### 4.1.4 Conclusions

The most striking result, which immediately became visible during testing, is that users do not understand the concept of different edit modes very well. They did not progress through the different edit modes as anticipated, but then again, this might be due to the specific tasks given to them. Still, the block editor seems to be most promising as about half of the users used the edit modes correctly eventually.

The difficulty of editing references also stands out, which seems to be inherent to wikitext. Even these users, who are well educated and have experience editing different kinds of syntaxes, had problems with this task. Edit modes do not help users understand complex wikitext like references.

It is not clear whether or not in-line editing works better than editing raw wikitext. However, it is hard to draw a definitive conclusion anyway, because of the highly biased group of users. The users in the control group were not visibly intimidated by the amount of wikitext they were confronted with, while Wikimedia research shows that most novice users are quite intimidated at first [21].

There were some noticeable improvements, such as the previewing inside the page. Users in the control group, and those who incidentally used the full editor edit mode, sometimes had problems previewing their edits. Because in-line editing allows for previewing inside the page as part of the edit process, there were no users having problems with previewing when using the functional or the block editor. However, most improvements were not directly related to in-line editing, but to the cleaner introduction. Users actually read the small amount of instructions, and they noticed the edit summary box more often.

---

<sup>4</sup>Category codes in MediaWiki are codes that are formatted like links, but which define in which categories the page is placed. There is no output at the place where these codes are placed, so they are easily mistaken for other links, in particular for references, as both are usually placed at the bottom of the page.

## 4.2 Testing the hybrid editor

In April 2011, researchers at the Greek Research and Technology Network conducted a usability experiment with the hybrid editor [7]. Unfortunately, there was no control group of users who used the original editor, or any of the older interfaces. They did, however, have a better group of participants. In total, 14 people took part in the experiment, half of which were inexperienced, novice users. The other half were experienced users, some of whom had actually edited wikis before.

### 4.2.1 Setup

The scenario was similar to the first usability experiment. Users were asked to perform a few tasks with an article of one of the suburbs of Athens, Nea Smirni. The tasks were divided in an easy and a hard part. The easy part consisted of correcting a typo, adding a sentence, deleting a paragraph, and changing the population count. The advanced part consisted of swapping two paragraphs, adding an external link, adding a reference, and moving an image. Unlike the first usability test, this interface also included an editing toolbar, as shown in Figure 3.1 on page 8. This toolbar was taken from the traditional wikitext editor of MediaWiki (Figure 2.2 on page 4). The participants could use the toolbar to add links, references, and so on. Also, there was a help section within the toolbar.

### 4.2.2 Analysis

The researchers looked at each task and identified the most common problems. Again, only a qualitative analysis was done because of the small group. The identification of common problems was based on recordings of the tests.<sup>5</sup>

After each test, the researchers showed the traditional wikitext editor of MediaWiki to the user, and asked which one they would prefer. They also asked what they would expect from an editor, and if they had any other comments.

### 4.2.3 Results

Most users completed the first part of the experiment in very little time. It seems that users were much more likely to find out how to edit paragraphs and sections than with the block editor. Sometimes it was necessary to give a subtle hint, such as: *“It is possible to edit a paragraph just like a sentence.”* This was necessary for the task where users had to delete an entire paragraph. After the hint, users had no problems completing this task. Editing of templates and images was also much less of a problem, as this is simply done by clicking, instead of having to select a certain edit mode.

In the advanced part, most of the problems that appeared concerned the wikitext syntax and not the interface. The interface-specific problems occurred mostly during the first stages of the advanced part. When asked to swap two paragraphs, most of the users still had trouble editing a section. Even an experienced user tried to drag and drop paragraphs, which makes sense, but was not implemented in the interface. Giving a small hint like the one before usually solved this problem.

When adding an external link, users struggled with the wikitext syntax. They sometimes used the external link button on the toolbar, but for most of the users this did not help.

References were even worse, as they have to be placed within the text instead of at the bottom. This is the exact same problem as with both the functional and block editor. Moving an image was also a syntax problem, the user had to replace the ‘right’ word by ‘left’, which is not what most users expected. Some were simply confused by the syntax while others tried right clicking on the image. There were also users who tried to edit the image itself by navigating to the image page.

---

<sup>5</sup>These recordings are not public. However, the transcripts are public and can be found in the original report [7], as well as online: [http://www.mediawiki.org/wiki/Extension:InlineEditor/Usability\\_study\\_2](http://www.mediawiki.org/wiki/Extension:InlineEditor/Usability_study_2).

When shown the original editor afterwards, all of the users agreed that the hybrid editor was better. When asked what they would expect, most of them mentioned a visual editor. Their comments also suggest that wikitext is still a problem, just as we saw with the first usability test. To quote from the report:

“They were also fascinated by the fact that it was quite easy to make quick changes like the ones included in the first part of the scenario. But the majority of their comments had to do with the wiki syntax. For example, most of the novice users mentioned that they were quite confused with the hyperlink syntax and the reference syntax.”

#### 4.2.4 Conclusions

While no control group was used to compare in-line editing with a traditional wikitext editor, when compared to the results of the usability testing by the Wikimedia Foundation [21], the results are quite positive. Users were not intimidated by large amounts of wikitext, and had no problems locating the elements they had to change. All of the users preferred the in-line editing interface when the traditional wikitext editor was shown, and they mentioned it was fascinating how easy it was to make quick changes.

The most significant problem of the interface is the editing of paragraphs and sections, which was still hard for some users to find out. In the first part of the test, most users found out that they could edit a paragraph, but in the second part it was necessary to edit an entire section, which most users did not find out. Perhaps the interface could give hints just like these given in the experiment. For example, after editing a sentence a popup could be shown indicating that it is also possible to edit paragraphs or sections. On the other hand, with typical usage editing entire sections is not required at all, so it might be a good idea not to give hints at all, and to have users find this out on their own.

Like we saw before, many problems are inherent to wikitext. While the toolbar that was added for this experiment should have been helpful to this end, the syntax was still hard to understand. It should be possible to give specific guidance when editing certain complex elements such as references, which we look at in section 6.2.



# Chapter 5

## Algorithms

The in-line editing interfaces presented in this thesis are particularly interesting because it is not necessary to make assumptions about the parser. They work without a formal grammar specification, and they work without having to change the existing parser.<sup>1</sup> In this chapter we first look at how visual editing is usually implemented, and then at how this can be adapted to build an in-line editing interface. Then we discuss building in-line editing interfaces when a syntax tree is not available, which most of the chapter is about. Finally, we look at the client side of in-line editing interfaces.

### 5.1 Visual editing

Usually, when parsing a certain language, a syntax tree is obtained, such as when using a parser based on a formal grammar. For example, take webpages that are written in HTML. When a webpage is parsed by the browser, a tree of objects is obtained, which corresponds to the original HTML. This is also known as the *Document Object Model* (DOM), which is the convention of representing objects in a webpage, so that interacting with these object is the same across browsers [25].

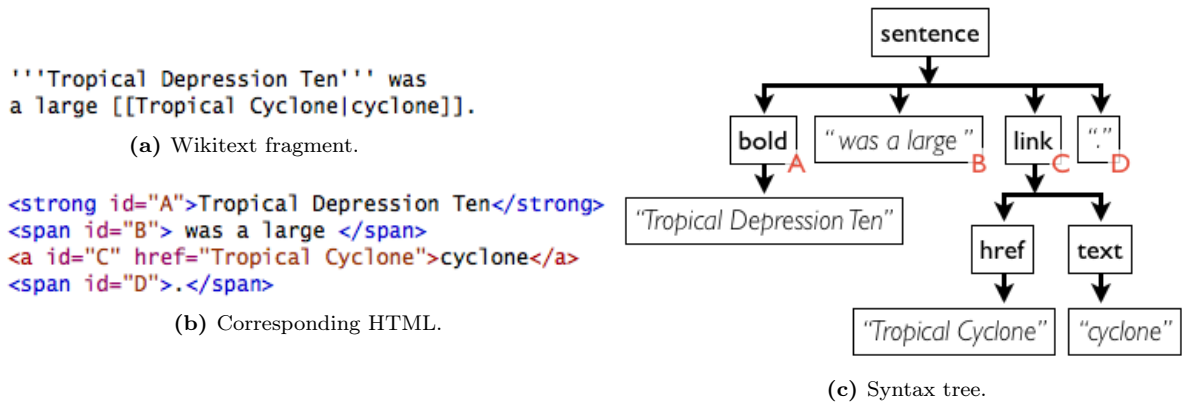
When parsing wikitext, a similar tree structure is obtained, but with an additional restriction. The tree has to contain all the information required to get back to the original wikitext, so-called *round-tripping*. This is because a visual editor is mostly used besides a traditional editor, to be able to edit raw wikitext for users who prefer this. When the wikitext of the page has been formatted in a certain way, for example for readability, we need to preserve this formatting when editing the page using a visual editor.

To build a visual editor, there are a few options. One of them is to send this wikitext tree structure to the browser, and render it using Javascript. Whenever the user makes a change, the tree structure can be updated, and the browser can render the page in its new state. Another option would be to send HTML to the browser, just as is done when viewing the page, but with additional information added to be able to do the round-trip back to wikitext. Additional information is also required to guide the visual editor when editing complex structures such as templates and tables. Sending this modified version of the page is an attractive approach, as most browsers have the ability to edit HTML natively, so that less code is required to do the rendering and editing using Javascript. This is the approach taken by the Wikia visual editor (see Figure 2.3 on page 6) [13]. However, this editor supports only a subset of MediaWiki wikitext, as it is hard to modify the existing parser to give enough information for clean round-tripping.

This is a problem that arises when building a visual editor: either a syntax tree must be available to the developer, such as generated by a parser based on a formal grammar specification, or otherwise the existing parser must be modified to add enough information to the output to make visual editing possible, for round-tripping back to wikitext and editing complex syntax.

---

<sup>1</sup>It is not entirely true that we do not make assumptions about the parser. We need to know which strings are left alone by the parser, and when making the optimisation of partial parsing, we need to make some changes in the parser to be able to handle dependencies. However, these assumptions and changes are minimal compared to what is needed when building a visual editor.



**Figure 5.1:** When using a parser based on a formal grammar specification, there is a direct correspondence between parts of the wikitext (a) and nodes in the syntax tree (c). For every node in the tree the original positions in the wikitext is known. When generating HTML it is possible to keep track of these nodes (b) and thus make elements clickable for an in-line editing interface.

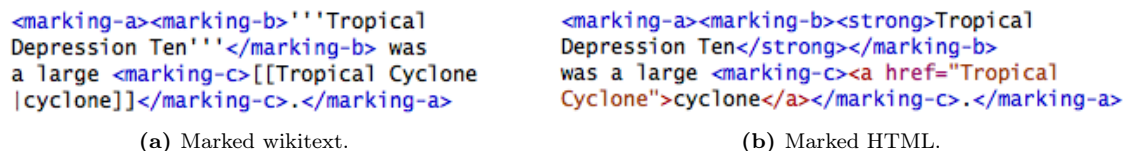
## 5.2 In-line editing

When building in-line editing interfaces we do not have to do round-tripping, as users edit wikitext itself. Instead, we need to know which elements in the final output of the parser correspond to which fragments of text in the original wikitext. For wikitext that can be described by a formal grammar, this is quite easy to know. In that case, the parser generates a syntax tree, which corresponds directly to the original wikitext (Figure 5.1). For every node in the syntax tree we know the exact start and end positions in the original text that it describes. When transforming the syntax tree to the output format, such as HTML, we can keep track of these positions, so that when clicking an element on the page we can edit the original wikitext.

However, this becomes problematic if there is no formal grammar specification for the wikitext syntax, as this could mean there is no syntax tree we could use to this end. We assume that we can only use an existing parser that takes wikitext as input and gives HTML as output. We treat the parser as a black box, which means we are only able to do transformations on the input and output. This way we do not depend on specific parser implementations or techniques. Although later on, we do make changes in the parser for an optimisation, in general this should not be required.

The basic idea is to add strings to the input that will survive parsing and can thus be found later in the output. For all practical purposes we assume that such *immutable strings* exist. We use immutable strings to make a correspondence between the wikitext and the output. This way, it is possible to surround certain fragments of wikitext that we want to make editable by immutable strings (Figure 5.2). We call such editable fragments of wikitext *markings*.

Now we do not have to define the entire grammar anymore, but only need to find parts of wikitext we



**Figure 5.2:** If we view the parser as a black box, we can only transform the input and output. By adding immutable strings to the input (a), we create a correspondence between the wikitext and the parser output (b). In this case, *marking-a* marks the entire paragraph, while the other immutable strings surround bold text (denoted with apostrophes) and links (denoted with double brackets).

want to make editable. This means we only have to make sure that we can detect different elements, instead of having to describe how they are actually parsed. For example, it is usually easy to detect a link, without knowing the exact syntax of the parameters of such a link.

Besides the advantage of detection instead of parsing, it also allows for graceful failing. If one would try to express an existing wikitext syntax in a formal grammar, it is easy to miss some edge cases. It is hard to know beforehand what the output in these cases would be, and how many pages there exist out there that would be parsed incorrectly when using this formal grammar.

When using the approach of adding markings and then leveraging the existing parser, it surely is possible that some structures are detected incorrectly. It is possible that the visual appearance of the page will be incorrect, but at least it is still possible to edit the original wikitext, for example by editing a high level element, such as a section. This is better than unpredictable behaviour, potentially making the entire page unreadable.

Our approach of transforming the wikitext consists of a few steps (Figure 5.3). First, we detect editable elements in the text, by using small modules that all detect particular elements. This gives us a list of markings, with their start and end positions. We then arrange these markings in a tree, to make it easier to do further transformations and checks.

Then we actually transform the wikitext by adding immutable strings to it, and we use the existing parser to parse the page. After that, we replace the immutable strings by HTML tags. We also check for misnestings, which might occur after template expansion.

After discussing these basic steps, we discuss an optimisation. When having parsed the page for the first time, the user will do edits after which the page will have to be parsed again. As this can be expensive, we look at a way to parse only a part of the page. We then consider the problem of resolving dependencies, which arises when doing this optimisation of partial parsing. Finally, we look at the client side of in-line editing interfaces.

## 5.3 Detecting elements

Instead of actually implementing the entire syntax of elements, we only need to be able to detect the boundaries correctly. Still, this can be harder than one would expect. Consider the case of detecting a sentence boundary:<sup>2</sup>

```
This is a sentence which {{some template|Another sentence.}}
```

 contains a template.

If we look for characters ending a sentence, such as the full stop, question mark and exclamation mark, then we would incorrectly find the full stop inside the template. The immutable strings would be added incorrectly:

```
<marking-a>This is a sentence which {{some template|Another sentence.</marking-a>  
<marking-b>}}
```

 contains a template.</marking-b>

---

<sup>2</sup>Even without wikitext syntax, finding the boundary of a sentence is quite difficult. Luckily, there are some documented methods for this, see: [http://en.wikipedia.org/wiki/Sentence\\_boundary\\_disambiguation](http://en.wikipedia.org/wiki/Sentence_boundary_disambiguation).

**Detecting elements** Modules are called to detect editable elements, in order to generate markings (section 5.3).

**Building a tree** The markings are organised in a tree structure, misnestings are discarded (section 5.4).

**Collapse markings** Remove some markings that share the same start and end points as other markings, based on metadata (also section 5.4).

**Marking wikitext** At start and end points of markings, immutable and unique strings are inserted in the original wikitext (section 5.5).

**Parse wikitext** The marked wikitext is parsed using the existing parser.

**Post processing** Immutable strings are replaced by HTML entities; misnestings are handled by removing markings (section 5.6).

**Figure 5.3:** Marking algorithm.

What would happen now is completely undefined, as templates can expand to anything. The markings could even simply disappear, if the template argument is not used. There are a few ways to counter this.

The first would be to actually allow this to happen, but to check for certain characters inside the sentence before adding the markings. In this case, we could check whether or not the sentences contain brackets, in which case we would discard the markings. While this seems crude, it is actually not a bad way to handle difficult cases, as only the sentence itself becomes un-editable. It would still be possible to edit the entire paragraph or section. It is also fast and easy to implement. Still, there are cases in which this approach may not give the desired result:

```
This is a sentence which {{some template|Another sentence. And another one.}}
contains a template.
```

The sentence “*And another one.*” would still be marked, as it contains no brackets. This is incorrect, as the sentence is inside a template. On the other hand, this can also be allowed to be able to edit sentences inside templates.

Another way would be to find all occurrences of templates in the text, and replace them by spaces before detecting sentences. If we take care to replace templates of a certain length by the same amount of spaces, then the start and end positions of the sentences will still be the same. This approach will detect this sentence correctly, but it is a bit more difficult to implement. For example, we have to take into account that templates can be nested.

A good way of organising the detection of elements in the code base, is by defining small modules to detect certain elements. This way, it is easy to test the behaviour of detecting certain elements, and to reuse correct detection of one element in the detection of other elements. For example, one module can detect templates to make them editable, but this detection can also be used in the correct detection of sentences, as in the example above.

Note that it is not strictly required that erroneous detection is fixed at this point. In the next stage all the markings are arranged in a tree, and it is possible to defer the detection of overlapping elements to that stage. We can then make sure that some elements get precedence over other elements, so that in the example above the sentence markings are removed because they have overlap with the template marking.

After the detection of different elements, they are arranged in a list of markings. Every marking needs to describe a start and end position in the original wikitext, where the immutable strings will have to be added. In addition, we can also define some metadata for each marking, such as the type of element. Each marking gets a unique identifier for finding the marking back in the output, by including it in the two immutable strings.

## 5.4 Building a tree

One way to organise markings is to arrange them in a tree. This means we do not allow partially overlapping markings, only full containment. As we are eventually going to replace the immutable strings in the output by HTML tags, correct nesting of the markings is required. Therefore, arranging them in a tree is a sensible choice.

Arranging markings in a tree is an easy enough process (Algorithm *build\_tree*). First, we sort the markings by position, ascending. When markings have the same start position, we take the longest marking first, then the second longest, etc. Now, when we walk the list of markings, every marking is either a child of the previous marking, or the child of one of the ancestors of the previous marking. We use this property to create the tree in a neat way.

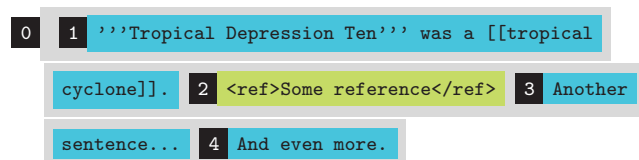


Figure 5.4: Wikitext markings for a simple paragraph.

**Algorithm** `build_tree( $M$ )`**Input:** List  $M$  of markings.**Output:** Tree of markings. $root \leftarrow$  new root markingsort  $M$  by start position (ascending), length (descending), level (descending) $node \leftarrow root$ **for all**  $m \in M$  **do**  **while**  $m.start \geq node.end$  **do**

{move one marking up in the tree}

 $node \leftarrow node.parent$   **end while**  **if**  $m.end \leq node.end \wedge m.start \geq node.last\_child.end$  **then**    add  $m$  as child to  $node$      $m.parent \leftarrow node$      $node \leftarrow m$   **else**

{invalid nesting, discard marking}

**end if****end for****return**  $root$ 

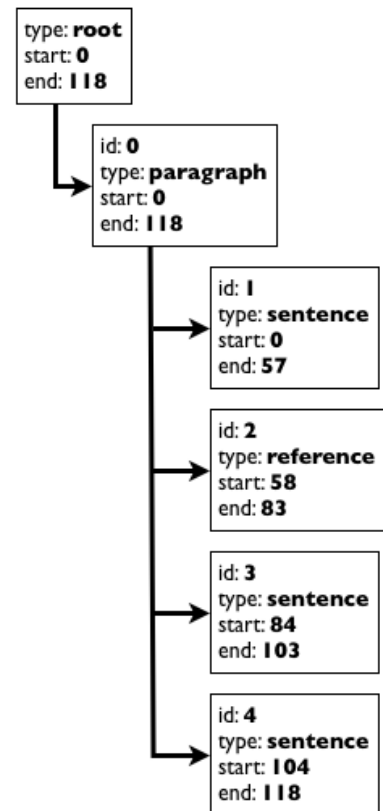
With this algorithm, every marking from the list is added as a node in the tree (Figure 5.5). The root marking is a special node, as it is not required that there is a marking spanning the entire page in the list of markings. For this special node it is not necessary to add immutable strings in the next stage.

Note that it is a useful to add the metadata property *level* to each marking. When there are two otherwise identical markings — i.e. with the same start and end positions — this property decides in which order the markings are placed. This way, a paragraph containing only a single sentence is marked correctly. The paragraph marking will contain the sentence marking, not the other way around.

It is also necessary that every type of marking has its own unique level. Otherwise, the order of two different markings with the same start and end positions is undefined. In section 5.7 we require that the order of markings is predictable, as we compare two ordered lists of markings with each other. Another option would be to use a stable sort algorithm, which makes sure identical markings will be kept in the order they were added by the detection modules.

There are some interesting operations that we can do on this tree of markings, which mostly have to do with how the hybrid editor will look like. The first one is the *collapsing* of markings. If we take the earlier mentioned example of a paragraph containing only one sentence, we can see that the paragraph marking is redundant. In the hybrid editor, the sentence itself will be clickable, while there will still be a bar to the left. Clicking this bar has the exact same effect as clicking the sentence itself. We can therefore safely hide this bar, cleaning the interface without losing functionality.

To do this, we walk through the tree in search for markings that have the same start and end positions as their parents. Then we retain the deepest markings, and remove (collapse) the ancestors that share their position. Some metadata can guide this algorithm, for example a *collapsible* property that determines whether an element can be removed this way.



**Figure 5.5:** Marking tree corresponding to a document with only the paragraph from Figure 5.4. The root node is a special node as it is not necessarily present in the list of detected markings.

There is another transformation we can do on the tree, specifically for the hybrid editor. For this one we look at the example of a sentence containing a template. With the functional editor, it is possible to select editing either the sentence or the template. With the hybrid editor, however, there is no way to explicitly state what kind of element to edit. Both elements will be highlighted by colouring the background. One solution would be to show some kind of context menu, so the user can select which element to edit. If we want to keep the interface more simple, another solution would be to remove the template marking, as the template can already be edited by clicking the sentence.

To remove the inner markings, we can add some metadata that states whether an element will have a bar on its left or whether the background will be coloured. We can then search for elements of the latter kind, and make sure that they are always leaves in the tree, simply by removing their children.

## 5.5 Marking wikitext

In order to run the existing parser, we first need the marked wikitext. This can be done by recursively walking through the tree (Algorithm *generate\_wikitext*). For each marking we can generate its corresponding wikitext, by generating the wikitext of its children and gluing their output together with the wikitext not surrounded by markings. Finally this wikitext needs to be wrapped in the immutable strings, unique for this marking.

**Algorithm** *generate\_wikitext*( $m, w$ )

**Input:** Marking  $m$  to generate wikitext for, wikitext document string  $w$

**Output:** String of generated wikitext

$pos \leftarrow m.start$

$s \leftarrow ""$

**for all**  $c \in m.children$  **do**

{note that these children are already sorted by start position}

$s \leftarrow s + w[pos \dots c.start]$

$s \leftarrow s + generate\_wikitext(c, w)$

$pos \leftarrow c.end$

**end for**

$s \leftarrow s + w[pos \dots m.end]$

**return**  $m.unique\_start\_string + s + m.unique\_end\_string$

We can use this function on the root of the tree and generate marked wikitext. One problem is choosing a good format for immutable strings. As wikitext usually leaves words alone, a string of letters is often a good choice. In MediaWiki, `div` tags are also left alone by the parser, and are thus suitable as immutable strings. One way or another, we have to look at the syntax and at the parser to determine a good format.

Another problem is that there is no way of knowing for sure that a string will be unique. Inspecting the wikitext is not enough, as a string can be hidden in a template. The only solution is to choose strings that are highly unlikely to be used, such as long strings with a certain prefix. To cover the rare case that such a string exists in the page after parsing, we can check if any immutable string occurs more than once, and then regenerate the marked wikitext and reparse the page, but this time with different strings.

After giving the marked wikitext as input to the existing parser, we get the HTML output from the parser with the immutable strings at the positions of editable elements. Next, we have to make a last transformation to obtain the final HTML document.

## 5.6 Post processing

To build the final HTML document from the parser output with the immutable strings in them, all we have to do is to replace these strings by HTML tags. In order to keep track of which tag belongs to which marking in the wikitext, we can add unique identifiers to the HTML tags.

In order to preserve some of the metadata attached to the elements, we can add this as `class` attributes in the HTML tags, or include it with some Javascript. Metadata can be used client side for rendering the page, for example to show a bar at the left for block elements and to highlight the background for inline elements, as we see in section 5.9. They can also be used to create specialised interfaces, as we discuss in section 6.2.

Another thing we can do during post processing is the detection and handling of misnesting. This may occur if a fragment of wikitext surrounded by immutable strings gets expanded by the parser to something that opens a HTML tag without closing it, or the other way around. For example, in MediaWiki the following construct is perfectly valid:

```
{{table open}}
{{table row|first cell|second cell}}
{{table close}}
```

In this case, the first template expands to only a table opening tag, and the last template will expand to a table closing tag. After marking the wikitext, it may look something like this:

```
<marking-a>{{table open}}</marking-a>
<marking-b>{{table row|first cell|second cell}}</marking-b>
<marking-c>{{table close}}</marking-c>
```

Then, this wikitext is parsed. If the strings are truly immutable, they will survive parsing. The result will look like this:

```
<marking-a><table></marking-a>
<marking-b><tr><td>first cell</td><td>second cell</td></tr></marking-b>
<marking-c></table></marking-c>
```

Then we replace the immutable strings by HTML tags, in order to produce an actual HTML document:

```
<div id="marking-a"><table></div>
<div id="marking-b"><tr><td>first cell</td><td>second cell</td></tr></div>
<div id="marking-c"></table></div>
```

This is obviously incorrect, as on the first line the `div` tag gets closed while the `table` tag is still open. On the third line a table gets closed while no table is open. Because of this misnesting the page will be displayed incorrectly.

There are, again, several ways to deal with this. The obvious way would be to check for each marking if misnesting occurs within that marking, and if so, to delete it. This would remove the *marking-a* and *marking-c* tags in the example before, but it leaves the incorrect *marking-b* tag as is.

A more aggressive method would be to detect misnesting only in certain types of markings, based on metadata, and remove all markings within such a high level marking when misnesting occurs. For example, in MediaWiki it should be possible to render a single section on its own, as the software already allows for previewing a single section. While the software does not enforce this restriction, it is a fair assumption to make that in most cases misnesting will only occur inside a section, not between different sections.

Therefore, we can remove all markings inside a section when misnesting occurs in that section. For this, we can step through the tags and immutable strings, looking for a misnesting. Note that it is better to do this before replacing immutable strings by HTML tags, as this allows for better matching of opening and closing tags. If we replace immutable strings by `div` tags, we would not know whether a closing `div` tag belongs to a marking or not, as the identifier and metadata are only stored in the opening tag. When doing this with the original immutable strings, this problem does not occur.

While this method is quite robust and easy to implement, it requires that such a property of independent parsing exists for certain elements. The most advanced option, for which this is not required, would be to construct a DOM (as discussed in section 5.1), containing both HTML tags and immutable strings, and then find out which immutable strings are invalid. HTML defines inline objects, which can be included anywhere, and block objects, which can be included only in other block objects. If we convert

the immutable strings to block objects, such as div tags, we can take this definition into account when determining which strings to remove.

To take it even further, we can use this DOM for the conversion of the immutable strings to HTML tags, so that we can obtain an output like this:

```
<table>
<tr id="marking-b"><td>first cell</td><td>second cell</td></tr>
</table>
```

In this case the second marking is collapsed into the HTML tag, which is a nicer solution than removing it altogether.

## 5.7 Partial parsing

Now that we have generated HTML output, an in-line editing interface can be shown to the user. The user can then make changes to an element, and click on preview. The page should be updated accordingly, so we can reparse the entire page after replacing the fragment of wikitext that has been edited. However, reparsing the entire page can be very expensive, so we should avoid it if possible.<sup>3</sup> It would be better if we could parse only a part of the page, which we call *partial parsing*.

One way to do this is to, again, use the property of some parsers that some elements can be parsed independently, such as sections in MediaWiki. We can find the closest ancestor of the element that was edited, that has this property. We then take its wikitext, and do all the marking, parsing, and post processing only on this part of the page. We substitute the node of this ancestor in the original tree by the newly obtained tree, and we only send the newly generated output to the client. This way, all the identifiers of markers on the client and server side still correspond.

Again, this is a robust and easy way of doing partial parsing in some wikis, but it is possible that there are wikis for which elements with such a property of independent parsing do not exist. It is also possible that this is simply not good enough, that it still takes too long to parse only such a part of the page. In this case, we can use a technique that uses the placement of the markings to determine which part of the page to parse.

Assume, for the moment, that we have *parsing locality*. That is, editing a bit of wikitext only changes the output at the corresponding location. This is usually, but certainly not always, a fair assumption to make. We look at how to solve the problem of dependencies in the next section.

If the output only changes where we made an edit, it is possible to only reparse this part of the page. We do have to generate markings for the entire page again, as we do not know what the consequences of an edit are for the placing of markings. Generally, there is some sort of *marking locality*, but we do not know to what extent. Actually, the marking locality can be considered a heuristic for the parsing locality. When some parts of the wikitext are detected differently, there is a good chance that it also will be parsed differently. We assume the marking to be much less expensive than parsing, so we will use this heuristic to determine which part we are going to reparse.

At first, the change will be made in the plain wikitext, and all the *previous markings* that have been placed before are moved accordingly (Figure 5.6a). Then, using the new wikitext, markings are generated again, the same way as before (Figure 5.6b). Now we have two sets of markings for the wikitext: the set generated using the wikitext after the edit, and the previous markings.

Now both sets are sorted in the same way as for building the tree, and both sets of markings are compared. When two markings coincide, the new one is replaced with the old one, to ensure the identifiers at the client side are still correct. When a mismatch occurs, the range where changes occur is updated to include this marking, and the new marking is not replaced (Figure 5.6c).

---

<sup>3</sup>An interesting anecdote is the case of the Wikipedia page of Michael Jackson. Because this page took a very long time to parse, all the visitors that went to the page when he died caused a cache stampede: <http://lists.wikimedia.org/pipermail/wikitech-1/2009-June/043593.html>.



Initially this *range of changes* is set to just include the marking that has been changed by the user, and it grows to include every marking that is different from the previous set of markings. Now the new set of markings has been made, it is arranged in a tree. By walking through this tree, we can find the smallest marking that contains the range of changes. In the worst case, this is the outermost marking which includes the entire page, in which case there is no performance gain. Usually though, this is just the marking that the user edited, or the one above that. By only rendering this one, the parsing time is reduced significantly, as is the bandwidth that is required.

Another use for the range of changes is to show the user what has changed. A yellow highlight is a recognisable indication of this (Figure 3.2 on page 9). Instead of finding the deepest marking that contains the range of changes, we find the highest markings that are contained in the range of changes. This can be done by using the marking that will be parsed, which we found before, and then walking all its children that overlap with the range of changes. We mark every child that is fully contained inside the range of changes, and for children that only have some overlap, we recursively walk through their children.

## 5.8 Handling dependencies

In many cases the parsing is *not* local. The most common example is that of references (Figure 5.7a). A reference is described in the wikitext at the position where only a number will be visible (Figure 5.7b), and the actual text is displayed at the bottom of the page (Figure 5.7c). Thus, when changing a reference, the section at the bottom should be updated to reflect this change. Other examples are parsing options in the wikitext, that specify how the page should be parsed. Obviously reparsing the entire page is required when such an option is changed.

There are a few ways to address this issue, but they all involve changes in the parser. We therefore have to let go of our principle of thinking of the parser as a black box.

The easiest way to solve this problem is to have the parser report each dependency it encounters. This works best for things like parser options: in this case the entire page should be reparsed, so when such a dependency is reported, we just interrupt the partial parsing and render the entire page instead.

```
The wind shear was expected to relent within
48 hours, prompting some forecast models to
suggestion the depression would eventually
attain hurricane status.<ref>http://www.nhc.noaa
.gov/archive/2005/dis/al102005.discus.002.shtml</ref>
```

(a) Reference in wikitext.

```
The wind shear was expected to relent within 48 hours,
prompting some forecast models to suggestion the
depression would eventually attain hurricane status.[4]
```

(b) Reference number.

4. <sup>^</sup> <http://www.nhc.noaa.gov/archive/2005/dis/al102005.discus.002.shtml> 

(c) Entry in the list of references.

**Figure 5.7:** A reference in the wikitext (a) resolves to only a number in the output (b), while the actual content of the reference is placed in a list (c).

```
1 '''Tropical Depression Ten'''
was a [[tropical cyclone]].
2 <ref>Some reference</ref>
3 Another sentence... {{Some
template}} 4 And even more.
```

(a) Original markings, which are moved for edited marking 3.

```
5 '''Tropical Depression Ten'''
was a [[tropical cyclone]].
6 <ref>Some reference</ref>
7 Another sentence... 8 {{Some
template}} 9 And even more.
```

(b) New markings.

```
1 '''Tropical Depression Ten'''
was a [[tropical cyclone]].
2 <ref>Some reference</ref>
> 7 Another sentence...
8 {{Some template}} < 4 And
even more.
```

(c) Replaced markings, with arrows denoting the range of changes.

**Figure 5.6:** An edit with partial parsing. First the original markings are moved (a), then new markings are generated (b), and finally coinciding markings are replaced (c) by comparing both the old and the new sorted lists with markings. After generating the new tree we find that the entire paragraph will be reparsed, because that is be the deepest node fully encapsulating the range of changes. Markings 7 and 8 get a yellow highlight to show they have been edited.

For more common dependencies such as references, this is not an option. It would defeat the entire purpose of partial parsing. A better option when encountering a reference during partial parsing, would be to add some client side Javascript code to the output, which greys out the reference list and adds a note saying it is out of date. A refresh button can be added to force the entire page to reparse when clicked. This is quite easy to implement and also makes the user aware of the fact that what just was edited somehow has something to do with the part of the page that has been greyed out.

The most advanced way of dealing with dependencies such as references is to actually resolve them. For this approach it is needed to modify the parser quite a bit, but doing this only for references should already catch the vast majority of the dependencies in wikitext. To know what dependencies to update, we should know which dependencies were previously included in the wikitext that is being parsed partially.

For this, the parser can output immutable strings itself at the position of the dependencies. We can then add these identifiers to our original tree of markings, in a post processing step. When doing a partial parse, we can tell the parser which references this part of the wikitext previously contained. In combination with the information of the previous parse, this can be used to update other parts of the page as well, by including some Javascript code to the output to do that.

For example, the references can be stored in a list while parsing. On a subsequent parse, the parser will be told which references were in that part of the wikitext before, so they can be removed from this list, and replaced by the ones that the parser finds. Then, the entire reference list at the bottom of the page can be replaced by the newly generated list by using some Javascript code that the parser adds to the output. Note that this approach is relatively easy to implement in MediaWiki as references are not part of the parser, but a separate extension.<sup>4</sup>

## 5.9 Browser

Most of this chapter is about creating the correspondence between the output of the parser and the original wikitext of the page. However, it is also relevant to discuss how to implement in-line editing interfaces client side, in the web browser.

To show the page, nothing new has to be done, as the output format of a wiki parser is already HTML. As we discussed earlier, the immutable strings have to be converted to HTML tags in a post processing stage, after fixing misnestings. When doing this, certain types of elements such as sentences and references can be converted to inline elements, such as `span` tags, while larger elements such as paragraphs and sections can be converted to block elements, such as `div` tags. To decide which tags to use for which elements, it is helpful to consult the HTML language reference [17].

When converting immutable strings to HTML tags, it is useful to include the unique identifier of the marking in the `id` attribute, for easy access using Javascript DOM functions [25]. Similarly, the type of marking can be included in the `class` attribute, to be able to display different elements in different ways, and even to build specialised interfaces for certain elements, as we discuss in section 6.2.

For the hybrid editor, displaying the bars on the left can be done by creating a border using CSS [2]. The effect when hovering this bar with the mouse, however, cannot be done with CSS, and requires some Javascript to add a `class` to the element when moving over the bar, and removing this again when moving the mouse away.

When clicking an element, a popup can be shown using Javascript. The text box containing wikitext can be made to automatically grow and shrink to fit the amount of text. With this method, it is not required to have scrollbars inside the text box, which can be confusing for novice users, and is more difficult to use on mobile devices.

The wikitext for a certain element can be found by making a query to the server, or by storing the wikitext for each element beforehand, by some Javascript generated by the server. This can be done

---

<sup>4</sup>The extension that manages the references is the Cite extension:  
<http://www.mediawiki.org/wiki/Extension:Cite/Cite.php>.

by walking the tree of markings, and add information about the markings in a large Javascript array, which is then included in the page. For example, the original wikitext of each marking can be sent to the browser. A more space efficient method is to include the original wikitext on the page with Javascript, and then store only the start and end positions in the array, thus avoiding duplicate wikitext.

When the user makes an edit, the browser sends a request to the server, which includes the new wikitext, and the identifier of the marking. The server also needs to know the previous state of the page, and the old tree of markings, to be able to substitute the new wikitext in the old page, and to do partial parsing. It is possible to store all this information in the browser, and send it to the server on each edit, however, this is not very efficient. A better way is to store this in a database, and only send the identifier of the row in the database to the client, as the browser has no use for all this server-side information anyway. This way, the browser only needs to send this identifier on subsequent edits. Old wikitext documents and marking trees can be removed when the user publishes the new page. They can also be stored for a certain amount of time, for example to be able to do undo operations after the page has been published. We look at the possibility of undo and redo operations in section 6.3.

The technology used for in-line editing interfaces in the browser is not particularly new, and is therefore compatible with a large range of web browsers. In fact, the current MediaWiki implementation is compatible with all modern browsers, and even with the most popular mobile browsers [7]. We think that this can be extended to include older browsers and most mobile browsers with relatively little effort, other than testing with all the different devices, operating systems, and browser versions.

## 5.10 Conclusions

This chapter shows how in-line editing interfaces can be built without a formal grammar, without making significant changes to the parser, and without having to port the parser to client side code. Unlike when building a visual editor, we can just use the existing parser, and do operations on the wikitext before parsing, and on the HTML output after parsing.

A lot of algorithms and suggestions are included in this chapter, which may or may not apply to certain situations. A basic implementation for MediaWiki has been included in Appendix B. In the next chapter we discuss how to improve the concept of in-line editing.

# Chapter 6

## Future work

While we have presented a few interesting interfaces built on a stable framework, there is a lot of room for improvement. In this chapter, we first look at some general interface enhancements that can be made, and then at specialised interfaces for certain editable elements. We see how edit transactions can be used for undo and redo operations, as well as for building live collaboration into in-line editing interfaces. We see that detection modules need to be improved in order to actually be able use these interfaces on real-life wikis. We consider a few ways of testing the robustness of the element detection, and we look at how in-line editing can be used at wikis already using a visual editor. Finally, we discuss the subject of changing the wikitext syntax, which we briefly mentioned in the introduction.

### 6.1 General interface enhancements

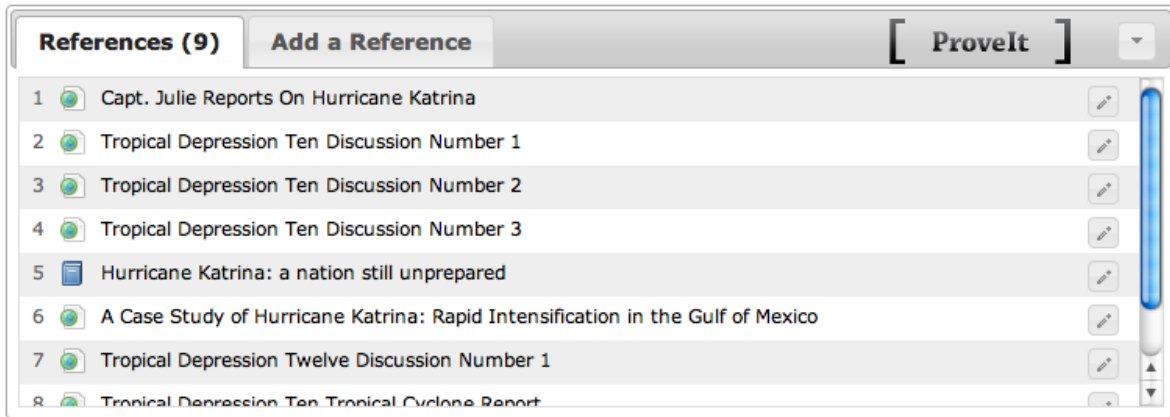
There are a few enhancements that could be made to the interface, that apply to all editable elements. For example, we already implemented the editing toolbar for the hybrid editor (Figure 2.2 on page 4). Other existing improvements to the editing text box could also be applied to the in-line editing interface, such as syntax highlighting and style mimicking (Figure 2.4 on page 6).

To help novice users understand the hybrid interface better, an interactive tutorial can be added, which guides the user through different elements that can be edited. Another way to help novice users is to detect frequently made mistakes, such as trying to select text. While to the user this makes perfect sense, in-line editing is not the same as a visual editor. Users may be helped by giving hints when frequently made mistakes are detected, the way hints were verbally given in the second usability experiment.

### 6.2 Specialised interfaces

Besides making general improvements to the interface, we can also improve the interface for certain types of elements. Such specialised editing interfaces have the potential to greatly enhance the editing experience. For example, we can implement live previewing of sentences. These contain a minimal amount of wikitext, a subset that can be implemented in client side code, to accurately show what the sentence would look like, while the user edits it. Of course a request would go to the server when the user clicks preview, and then the sentence will be actually parsed.

For templates we can show the instructions associated with the template. Usually, when navigating to the template page, there is information on how to use it, what the different options are. This can be shown by the template detection module, whenever a user edits a single template. Template parameters can also be made easier to edit, by allowing users to edit individual parameters in text boxes, instead of having to work with the entire template at once.



**Figure 6.1:** The ProveIt tool for managing references.

Another great example is ProveIt (Figure 6.1), a tool for managing references made by researchers at the Georgia Institute of Technology [10]. This can be integrated into a reference detection module. By doing this, the reference detection can be greatly enhanced, as they have already developed a good way of finding references. Also, their interface can be shown whenever a reference or reference list is being edited.

The most interesting examples, however, could be the ones that no-one has thought of yet. It would be interesting to see whether users and developers can come up with original ideas that would be hard to build right now, but that would be easy to build with the modular design of this editing interface.

### 6.3 Edit transactions

Because the user explicitly clicks a preview button, we have well-defined transactions, i.e. atomic operations on the document, which has several advantages. First, undo and redo functionality is trivial. Having relatively large transactions completely avoids the problem of not knowing how much to undo, as is usually the case with either plain text or visual editing. We already implemented this, with buttons placed next to the edit modes (Figure 3.4 on page 10 and Figure 3.6 on page 10). For the hybrid interface these buttons have been placed in a bar with advanced edit options for power users.

But edit transactions are even more powerful, as we can implement a collaborative interface using this. By using a long-lived HTTP connection (i.e. Comet [5]), we can wait for the server until a new transaction is ready, and handle it in the exact same way as transactions of the current user are handled. This way, multiple people can work on the same document together, and publish it when they are done. By adding a user identifier to each transaction, we can also show highlights of different colours, corresponding to each user. Combining this with a chat interface, this can bring wiki collaboration to a whole new level.

### 6.4 Detection modules

As the focus of this thesis is to evaluate in-line editing interfaces, and to determine whether or not this is suitable for wikis without a formal grammar specification, we have only worked with very basic detection modules. In order to run an in-line editing interface on a live site, it is necessary to vastly improve the detection of editable elements.

To this end it is useful to let go of the abstraction of the parser as a black box, and to look at how the parser may help to do correct detection. For example, in MediaWiki parsing consists of a preprocessing stage, in which templates are found and expanded. The information from this stage could be used for reliable template detection.

## 6.5 Robustness metrics

Because by definition parsing should be invariant to marking, we can quantitatively measure the robustness of the marking process. The way to do this is to apply the markings, parse the wikitext, and then remove the markings. The result should be the same as when only parsing the page. By taking a sample of pages of live wikis, problems during marking can be traced this way, without bothering users.

In a similar way, we can test whether or not certain markings can be parsed independently. By comparing the output of a marking within the page with the output generated when parsed on its own, we can check this behaviour. Of course, testing this feature is more time consuming than testing the marking invariant.

Another metric that is even more time consuming, is testing whether marking locality is a good heuristic for parsing locality. This can be done by changing some wikitext, and checking whether the marking differences indeed correspond with the differences generated by the parser. As we usually do not know what kind of change may invoke a dependency, it is hard to know what to change in the wikitext. This can be done randomly, by making manual test cases, or by using older revisions of existing pages on Wikipedia or other wikis.

Acquisition of hard data like this is important to convince the community of the quality of the editor. For example, the MediaWiki community has seen many attempts to create a new parser [11], so they are quite skeptical of new developments that involve parsing wikitext in a different way. The acceptance of such an interface among developers and users can be helped by backing arguments with data like this.

## 6.6 In-line editing and visual editing

Further research has to show the effect of different interfaces on goals like increased editor retention, better usability for novice and power users, better initial response to the interface, speed of learning to do more complex tasks, and so on. It would be interesting to compare in-line editing with visual editing for different tasks, and with users with different experience.

## 6.7 Changing the wikitext syntax

In the introduction we mentioned that usability of wikis has to be improved either by changing the editing interface, or by improving the wikitext syntax. While this thesis focuses on the first, changing the syntax is also an interesting possibility. Recent efforts by both academics and wiki developers to improve and standardise the wikitext syntax are actually quite promising [20]. They propose an unambiguous syntax to be implemented on all wikis. For existing wikis, they propose to allow users to edit in either the old or the new wikitext language. In our opinion, this approach of standardising wikitext should be further investigated.

## Chapter 7

# Conclusions

This entire research would not have been necessary if a visual editor could be used for all wikis, as this is a convenient way to edit wiki pages, and users are used to it. However, in reality there are wikis for which this is not possible because no formal grammar specification exists. These wikis still show the user one huge text box, which is not very user friendly. Our research shows that for these wikis in-line editing is a vast improvement. Still, wikis that do not have these limitations may also benefit from in-line editing interfaces, as they are convenient for power users who still want to edit raw wikitext.

Of the three interfaces we developed and tested, the hybrid editor is clearly the best. While novice users still struggle to find out how it works when starting to use the interface, this is to be expected with a new paradigm. Once users find out how to use it, it proves to be a more powerful interface than the traditional wikitext editor.

The method of generating in-line editing interfaces leverages the existing parser, instead of requiring a formal grammar. This may be favourable instead of writing such a grammar, making changes in the existing parser, or porting this parser to client side code. Still, the algorithm has its limitations, such as having to remove editable elements when misnesting occurs, and having to put extra work into resolving dependencies.

There are many opportunities for further work, such as incorporating syntax highlighting, style mimicking, specialised interfaces for specific editable elements, and better detection modules. It could even be feasible to implement live collaborative editing using in-line editing interfaces. Also, to show the robustness of the algorithm and of the detection modules, some metrics on real data can be calculated.

# Acknowledgements

I would like to thank my supervisors, Leonie Bosveld-de Smet and Gerard R. Renardel de Lavalette, for their helpful discussions and reviews; the MediaWiki community and the Wikimedia Foundation — especially Roan Kattouw — for their input, and their invitation to the MediaWiki Hack-A-Ton in Washington D.C.; the participants in the first usability experiment for their time and permission to publish the recordings; Dimitris Mitropoulos, Dimitris Meimaris and Panos Louridas at the Greek Research and Technology Network for their great contributions to the extension and for doing the second usability experiment; and finally Arno van der Vegt, Daan Davidsz, Ben van Os, Mark IJbema, Femke Hoornveld, and my parents for their invaluable suggestions and moral support.



# Bibliography

- [1] Alexa. Top sites. <http://www.alexa.com/topsites>. Retrieved on June 4th 2011.
- [2] B. Bos, T. Çelik, I. Hickson, and H. W. Lie. Cascading style sheets level 2 revision 1 (css 2.1) specification. World Wide Web Consortium, Recommendation, July 2007.
- [3] British Broadcasting Corporation. Jimmy wales says wikipedia too complicated for many. <http://www.bbc.co.uk/news/technology-12171977>, January 2011. Retrieved on July 6th 2011.
- [4] B. R. Cowan, L. Vigentini, and M. A. Jack. Exploring the effects of experience on wiki anxiety and wiki usability: an online study. In *Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology*, BCS-HCI '09, pages 175–183, Swinton, UK, UK, 2009. British Computer Society.
- [5] D. Crane. *Comet and Reverse Ajax: the Next-Generation Ajax 2.0*. APress, Berkeley, 2008.
- [6] A. Désilets, S. Paquet, and N. G. Vinson. Are wikis usable? In *Proceedings of the 2005 international symposium on Wikis*, WikiSym '05, pages 3–15, New York, NY, USA, 2005. ACM.
- [7] Greek Research and Technology Network. Wysiwyg support for wikipedia using the inline editor, 2011.
- [8] H. Kuusela and P. Paul. A comparison of concurrent and retrospective verbal protocol analysis. *American Journal of Psychology*, 113(3):387–404, 2000.
- [9] L. Lamport. Document production: Visual or logical. *Notices of the American Mathematical Society*, pages 621–624, June 1987.
- [10] K. Luther, M. Flaschen, A. Forte, C. Jordan, and A. Bruckman. Proveit: a new tool for supporting citation in mediawiki. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, WikiSym '09, page 43:1, New York, NY, USA, 2009. ACM.
- [11] MediaWiki. Alternative parsers. [http://www.mediawiki.org/w/index.php?title=Alternative\\_parsers&oldid=414395](http://www.mediawiki.org/w/index.php?title=Alternative_parsers&oldid=414395), July 2011.
- [12] MediaWiki. Future. <http://www.mediawiki.org/w/index.php?title=Future&oldid=407655>, June 2011.
- [13] MediaWiki. Wikia reverse parser. [http://www.mediawiki.org/w/index.php?title=Future/Wikia\\_Reverse\\_Parser&oldid=410991](http://www.mediawiki.org/w/index.php?title=Future/Wikia_Reverse_Parser&oldid=410991), June 2011.
- [14] MediaWiki. Wysiwyg editor. [http://www.mediawiki.org/w/index.php?title=WYSIWYG\\_editor&oldid=415380](http://www.mediawiki.org/w/index.php?title=WYSIWYG_editor&oldid=415380), July 2011.
- [15] OECD. *Participative Web and User-Created Content*. OECD Publishing, 2007.
- [16] F. Ortega. *Wikipedia: A Quantitative Analysis*. PhD thesis, Universidad Rey Juan Carlos, 2009.
- [17] D. Raggett, A. Le Hors, and I. Jacobs. Html 4.01 specification. World Wide Web Consortium, Recommendation, December 1999.

- [18] M. J. Rees. User interfaces for lightweight in-line editing of web pages. In *Proceedings of the First Australasian User Interface Conference*, pages 88–94, Washington, DC, USA, 2000. IEEE Computer Society.
- [19] C. Sauer. What you see is wiki - questioning wysiwyg in the internet age. In *Proceedings of Wikimania 2006 - The Second International Wikimedia Conference*, 2006. Online proceedings.
- [20] C. Sauer, C. Smith, and T. Benz. Wikicreole: a common wiki markup. In *Proceedings of the 2007 international symposium on Wikis, WikiSym '07*, pages 131–142, New York, NY, USA, 2007. ACM.
- [21] Wikimedia Foundation. Wikipedia usability initiative. [http://usability.wikimedia.org/w/index.php?title=Wikipedia\\_Usability\\_Initiative&oldid=9338](http://usability.wikimedia.org/w/index.php?title=Wikipedia_Usability_Initiative&oldid=9338), December 2010.
- [22] Wikimedia Foundation. Strategic plan. [http://strategy.wikimedia.org/wiki/File:WMF\\_StrategicPlan2011\\_spreads.pdf](http://strategy.wikimedia.org/wiki/File:WMF_StrategicPlan2011_spreads.pdf), February 2011.
- [23] Wikipedia. Systemic bias. [http://en.wikipedia.org/w/index.php?title=Wikipedia:Systemic\\_bias&oldid=425280033](http://en.wikipedia.org/w/index.php?title=Wikipedia:Systemic_bias&oldid=425280033), April 2011.
- [24] Wikipedia. Wiki. <http://en.wikipedia.org/w/index.php?title=Wiki&oldid=436161033>, June 2011.
- [25] L. Wood, A. Le Hors, V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, G. T. Nicol, J. Robie, R. Sutor, and C. Wilson. Document object model (DOM) level 1 specification (second edition). World Wide Web Consortium, Working Draft, September 2000.
- [26] E. Zachte. Wikipedia statistics. <http://stats.wikimedia.org>. Retrieved on July 6th 2011.

# Appendix A

## Usability testing

In this appendix we look at some details of the first usability experiment, as discussed in section 3.4. In section A.1 the original script that was used is included. Then a short form which was given to the participants is shown, together with the results. Finally, we look at the transcripts of the videos.

### A.1 Script

This script is based on a script published by the Wikimedia Foundation [21].

#### A.1.1 Introduction and setup

- Thank you for coming in.
- I'll be recording both your screen and our audio conversation. These recordings are going to be posted on the internet and available for those interested in this research to view at least part of them. If you don't want this, we'll stop now and continue with the next participant.
- We're collecting feedback on Wikipedia. Basically I'll ask you some questions and then watch you use the website.
- This will take about 15 minutes.
- Afterwards, I'll ask you to fill out a form with a few questions.
- Do you have any questions so far?

#### A.1.2 Explanation

- Your job is really easy, you just have to be yourself and act as you naturally would. I don't actually work directly for Wikipedia, I'm just collecting feedback. As you interact with the site, please be honest with your positive and negative thoughts. There's no right or wrong answers, and nothing you say will hurt my feelings.
- You are not being tested, the Wikipedia pages are being tested.
- Tell me a little about yourself.
- Can you tell me what you know now about the process of editing?
- There's one thing that I would like you to do differently. Please think aloud as you use the web site today. For example, if you are reading something read it out loud and feel free to say anything that comes to your mind as you read — for example “that's interesting” or “what in the world are

they talking about”? Also, talk out loud while you’re doing a task, so I can understand what you are thinking and doing. And, if you get to a point where you would naturally stop working, let me know.

### A.1.3 Tasks

- We’ll be using a copy of Wikipedia, sort of, with only the page Groningen.
- In the first paragraph there is a typo. Can you fix that typo?
- If you wanted to add a sentence or paragraph of info to this page, how would you do that? Go ahead and show me. (pre-fab sentence to give user if they don’t know what to add: “Er gaat niets boven Groningen!”)
- (if user doesn’t mention References) Did you see the section about references? Can you add a reference to the city site (<http://www.groningen.nl>)?<sup>1</sup>
- (if user doesn’t mention External Links) Did you see the section about external links? Can you add a link to the university site (<http://www.rug.nl>)?
- Let’s say you just moved to Groningen and wanted to change the population field to be one greater. How would you do that? (Point out where this information is on the actual entry if necessary.)
- (if time permits) Do you see this image of the Martinitoren? Can you change it so it’s on the left of the screen?<sup>2</sup>
- (if time permits) Can you swap these two paragraphs?<sup>2</sup>

---

<sup>1</sup>After doing the test with the first two participants, adding a reference was moved after adding an external link, because of the difficulty.

<sup>2</sup>The image moving task was never done, and the paragraph swapping task only once; therefore they are omitted from analysis.

## A.2 Form

As the time for the tests was limited to only 15 minutes, the participant was asked to fill out a short form afterwards.

### A.2.1 Questions

The form contained the following questions:

- First and last name
- Age
- Gender
- How often do you use Wikipedia?
- How often have you edited before?
- Do you have a username on Wikipedia? If so, what is it?
- Do you have any suggestions on how to improve the editing of Wikipedia?

### A.2.2 Results

The results of these questions are included in Table A.1. The interface the participants used is also included in this table. For reasons of privacy the age of the participants is not included in this table. The ages ranged from 18 to 29 years.

**Table A.1:** Usability test 1 form results

#	Gender	Education <sup>b</sup>	Wikipedia viewing	Wikipedia editing	Interface <sup>a</sup>	Suggestions
01	F	MA	almost every day	never	O	edit button also besides first section
02	F	MA	-	never	B	full editor; WYSIWYG
03	M	MA	once a week	never	F	making the adding/managing of references easier
04	M	MA	once a day	never	B	-
05	M	MA	every week	never	F	better terminology; less content; less guidance / text to read
06	M	BA	few times a week	never	F	-
07	M	BA	once a day	never	B	-
08	M	BA	2 times a month	1 time	O	syntax highlighting
09	M	BA	often	never	B	edit mode bar larger; per sentence editing is great; full editor isn't that bad when you use it often
10	M	BA	5 times a week	1 time	B	making the adding of references easier

<sup>a</sup> *F*: functional editor; *B*: block editor; *O*: original editor.

<sup>b</sup> *MA*: master's student; *BA*: bachelor's student.

## A.3 Transcripts

These transcripts are based on the recordings of the experiment. These recordings are publicly available online.<sup>3</sup> As in the videos Dutch is spoken, these transcripts are convenient for an international audience.

### A.3.1 Participant 01

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
female	master's student	almost every day	never	original editor

**Adding a sentence** She first searches for an edit link, and finds a section edit link, and she soon finds out that she's not editing the correct section. She's confused that there's no such link for the first section. I had to direct her to the tab at the top.

Scans the entire edit page, including all the warnings and other information.

Now she quickly identifies the sentence to add and does this correctly. Wants to preview the page and is confused which button to use. Manages to get it working and saves the page, without filling in an edit summary.

**Adding reference** Clicks the section edit link and finds the result confusing. Confuses category links with references, and misses the references template. Tries to add a reference simply by adding a url at the bottom, and finds that this does not work properly.

Tries the edit toolbar for advanced functions, and finds the insert a link button, mistakes this for a reference. Tries to use the ordered list function. Finds that this gives a different list.

I give the hint that the reference might be found in the text. Now she quickly edits the section and copy-pastes a reference, which works correctly. At first, she doesn't recognise her addition, and tries to fix her old addition. I give the hint that she's done it correctly.

**Adding an external link** She quickly and correctly adds an external link, but once again omits an edit summary.

**Changing population count** Changing the count in the text is no problem for her. When changing the count in the template she clicks a label and is taken to an empty page. She gives up.

I tell her that she can click the top tab again. She recognises the number and changes it correctly.

**Swapping paragraphs** She uses a section edit link again, and quickly swaps the paragraphs by dragging and dropping.

---

<sup>3</sup>The material is available for download here:  
[http://commons.wikimedia.org/wiki/Category:Sentence-level\\_editing](http://commons.wikimedia.org/wiki/Category:Sentence-level_editing).

### A.3.2 Participant 02

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
female	master's student	-	never	block editor

**Fixing typo** She first scans the entire page and clicks a section edit link. She immediately enters the edit summary, and she notices the examples below the text box.

She notes that the editing is through direct manipulation, and also says that the syntax isn't really great. I tell her to save the page, and goes to the bottom. When finding nothing there, she quickly finds the publish button.

**Adding a sentence** She finds a section where to add the sentence and clicks the edit section button (but with this interface the entire page is edited).

Enters an edit summary, and notes that it's weird that you have to enter it before actually editing, like you have done it already. She also notes that having an actual cursor would be better. She plays around with the sentence and then saves the page.

**Adding a reference** She notices that it's weird that clicking a section edit button edits the entire page. She tries to add a reference in sentence mode, and finds out that it's impossible to find the reference code this way.

Being stuck, she defers to entering the edit summary. Now she says that she would normally stop about now, and perhaps google this information. I tell her this is okay, and she heads to google to find information. On a help page she finds the correct syntax for references. However, she doesn't really understand the syntax and explains her expectation, which is some sort of HTML-like syntax. She states that she would definitely stop now.

We decide to continue, and she tries to add a reference with the syntax found online. She finds that this kind of works, but as she forgot to add 'http://', the link isn't automatically created and she is not satisfied.

**Adding an external link** She finds that in sentence edit mode, the list isn't highlighted. She goes back and forth on the page, and doesn't try to switch to another edit mode. She tries to google it again, but obviously doesn't find anything as this is a new interface.

I give her the hint to switch to a different edit mode. First she tries the full editor, and as she is quite familiar with editing syntax like this, she quickly finds the correct section and copies the format of another link. She notices that the save page button is a bit far away from the input field, but as there isn't anything else she clicks it anyway. She doesn't notice the edit summary field here.

**Changing population count** She clicks a section edit link again, and fills in an edit summary. She first edits a sentence, and notices that editing the template right away doesn't work, so she first saves the page. She then proceeds to the full editor again, and finds the correct field and changes it. She tries the preview tab, and is confused as she can't find the publish button right way. Then she finds the save button and notices the inconsistency with the publish button seen before.

### A.3.3 Participant 03

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
male	master's student	once a week	never	functional editor

**Fixing typo** Clicks the edit tab at the top. Reads the instructions and enters the edit summary. Expects the publish button at the bottom of the page. Reads the examples of wiki syntax. Scrolls through the page itself, and clicks on a sentence. Notices the wiki syntax and explains that this is probably some kind of markup or links. Corrects the typo. He notices that the Dutch translation of preview is a bit weird, and he would expect a save button here.

He clicks the preview button and likes the fact that the sentence is highlighted in yellow. He now clicks the publish button which he noticed earlier.

**Adding a sentence** Clicks the edit tab at the top again and enters the edit summary. He thinks a second about how to do this, but then clicks a sentence and adds his sentence at the end.

**Adding an external link** He now clicks the section edit button, notices that the entire page is edited. Enters an edit summary. Finds that it's impossible to edit the list using in sentence mode. Notices the edit modes bar. Wants to select the references mode. Is confused as to which mode to select.

I tell him he can just try something out. Thinks a bit about which mode to select. Concludes that it must be some kind of list and selects the list mode. Finds that this does work. Selects sentences again to check if it's something different, and then selects the list mode again. He clicks on the correct list and copies the syntax from an earlier list item.

He incorrectly copies the syntax and when clicking preview he finds out his link doesn't have a label. He tries to edit again and notices his mistake and fixes it.

**Adding a reference** He remarks that it doesn't matter which edit button to click, which isn't that logical. Selects the reference edit mode. Scrolls to the external links section, and completely misses the fact that references in the text are highlighted. Proceeds to read the help that shows the syntax of a reference.

He switches back to the sentences edit mode and scrolls a bit. He then selects the list mode again and explains that he feels he should use that. He then switches to references again and tells me that references aren't highlighted. He explains how to add a reference using a tag, and then selects the list edit mode again. He tries to edit external links again, and I tell him I want a reference, not an external link.

He tries the reference edit mode again, but notices that the reference list at the bottom of the page can't be edited this way. He clicks full editor but misses the fact you have to click a link. I ask him if he'd have stopped by now. He tells me that normally he likes to try things out, and search for it. I tell him that it is okay. He notices the more information link provided by the reference edit mode, but doesn't click it. He struggles a bit with the explanation, and finally clicks the link. He reads a bit about references, but still doesn't know what to do with it. We decide to skip this assignment for now.

**Changing population count** He clicks the edit tab at the top, and enters an edit summary. The population count in the text is no problem, but he struggles with editing the info box. He tries the media edit mode, and finds out that this doesn't work. He tries the template edit mode, and finds that in this mode the info box is highlighted. He clicks the info box, fixes it, and we notice a few bugs here.



### A.3.4 Participant 04

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
male	master's student	once a day	never	block editor

**Fixing a typo** Clicks the edit tab at the top. Reads the introduction but does not enter the edit summary. Quickly scrolls to the sentence, clicks it and edits it. He objects to the Dutch translation to 'preview'. He scrolls to the bottom to find a save button but finds nothing and scrolls back. He then enters an edit summary and clicks publish.

**Editing a sentence** He searches for a spot to edit something. Clicks the edit tab again. Clicks a sentence and adds a sentence at the end. He notices that the reference has been moved and he doesn't like this and tries to fix this, but thinks this is not possible. He clicks the back button in the browser and tries to edit again. This time he clicks the sentence after the reference and adds the sentence in front. He notices that he might have given up now when doing this at home. He forgets an edit summary but as the system does not complain he doesn't mind.

**Adding a reference** He looks at the different edit mode but doesn't think these are useful: "paragraphs, sections, no no no." I tell him he can Google if he'd like, but he likes to try in the editor first. He clicks on a reference but nothing happens. He scrolls a bit and eventually defers to Google. He finds a page about reference templates. This doesn't really help, and he tells me it isn't easy at all.

I give him the hint to try a different edit mode. He tries out the paragraph mode, and notices a reference tag. He copies this tag, but edits it incorrectly. He notices that it didn't work out correctly but we decide to leave it as-is.

**Changing population count** He doesn't have any problems with editing the sentence. For the template he switches to section edit mode, and this also works out just fine.

### A.3.5 Participant 05

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
male	master's student	every week	never	functional editor

**Fixing a typo** He clicks the edit tab at the top and reads the instructions. He doesn't fill in an edit summary just yet. He clicks on the sentence and fixes the typo. Tells me he would expect a button 'change' instead of 'preview'. He isn't sure whether or not the page has been edited now. He scrolls down and doesn't find a confirmation button, and assumes that the page has been changed.

He clicks back to the read page and notices that the page hasn't been edited. He tells me it would be possible he would have stopped by now, but he navigates to the edit page again and immediately notices the huge publish button. He reads the introduction more carefully now. He tells me that he gets too much information, which is the reason that he didn't read it this carefully before. He edits the page again, fills in an edit summary and publishes the page.

**Adding a sentence** Again, he clicks the edit tab at the top. He is a bit confused how to do this, but then clicks a sentence and adds his new sentence at the end. He tells me he isn't sure how the syntax works, and assumes that the system will take care of this. He edits the sentence, notices he missed a space, fixes this, and publishes the page. I tell him he doesn't have to add an edit summary this time.

**Changing population count** Editing the sentence isn't a problem. He isn't really sure whether or not editing the info box will be successful, so he first saves the page. On the view page he clicks on a link in the info box which takes him to an empty page. I give him the hint he might want to go back.

He heads to the edit page again. He scrolls through the entire page looking for clues as to how to edit this. We decide to stop now.

### A.3.6 Participant 06

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
male	bachelor's student	few times a week	never	functional editor

**Fixing a typo** He clicks a section edit link and is taken to the edit page of the entire page. He clicks the sentence, immediately fixes the typo, scrolls the entire page to find some kind of save button but completely misses the huge publish button. Eventually he finds this end publishes the page without entering an edit summary.

**Adding a sentence** He clicks the sentence after which he'd like to add the sentence and adds it. He publishes the page again.

**Adding an external link** He clicks the section edit link next to the external link section. He reads some of the instructions on syntax, and now scrolls to the section again. He notices that the list isn't clickable. He searches the page for a button that allows him to add an external link. He uses CTRL+F to search on the page. He now notices the different edit modes and clicks the reference mode.

I tell him we don't have to add a reference right now. He clicks one of the external links, but nothing happens. He struggles with selecting the right edit mode and clicks on a more information link. He tells me he would've stopped by now.

Now I give him the hint to try out different edit modes. He clicks a few of the different edit options. He ends up using the full editor, and doesn't have too much trouble to identify the external links section. Now he correctly copies the syntax of another link. He selects the preview tab and checks if the edit was successful. He also checks the edits tab and then saves the page.

**Adding a reference** I tell him how a reference works and ask him to add one. He quickly finds the full editor again and copies an existing reference tag. He correctly edits the tag, checks the preview and edit tabs and saves the page.

**Changing population count** He switches to edit mode and clicks a link in the info box, but nothing happens. He tells me he hopes that when changing one of the values, the other one changes automatically. He uses the sentence edit mode to switch the one in the text. He then proceeds to the full editor, losing his changes, but correctly changes the value in the info box. When saving, he notices that the earlier changed sentence hasn't been saved correctly, so he tries to edit the page again and now only changes the sentence and saves it correctly.

### A.3.7 Participant 07

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
male	bachelor's student	once a day	never	block editor

**Fixing a typo** He clicks the edit tab at the top. He clicks the sentence, fixes the typo, hits the preview button, and publishes the page without entering an edit summary. I ask him if this is what he expected, and he tells me it is.

**Adding a sentence** He heads to the edit page again, and quickly clicks on different edit modes. He selects the paragraph edit modes and clicks a paragraph to find out how this works. At the end of the paragraph he adds the sentence. After checking his sentence for spelling mistakes, he publishes the page. In a glimpse he notices he didn't enter an edit summary and asks me if he should've filled this.

**Adding an external link** On the edit page, he scrolls a bit to find the correct section. He selects the full editor option and clicks the link. He finds the correct section and copies the syntax of a another link. He tries to find the save button and finds it after some looking.

**Adding a reference** I ask him whether or not he knows how references generally works, and he tells me correctly how it works. He moves to the full editor and he tells me he thinks it's not as clear, but more powerful than the different edit options. He copies the syntax of a reference tag. He saves the page and sees that the reference isn't quite correct but we accept how it looks now.

**Changing population count** On the edit page he selects the section edit mode again. He also notices what he should've done when adding the reference. He then changes the count in the info box, and after that he changes the count in the sentence. He publishes the page.

### A.3.8 Participant 08

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
male	bachelor's student	2 times a month	1 time	original editor

**Fixing a typo** Scrolls through the page and clicks on a section edit link, and tells me he expects there is a similar link at the top. Eventually he finds the tab at the top and quickly finds the typo and fixes it. He doesn't enter an edit summary and saves the page.

**Adding a sentence** He clicks the edit tab at the top, and scrolls through the text to find the end of the first paragraph. He clicks back to the page to check what the end of the paragraph is, clicks back to the edit mode and adds the sentence.

**Adding an external link** He clicks the correct section edit link and copies the syntax of the existing list of links.

**Adding a reference** Again, he clicks the edit tab at the top. He learns about the syntax by clicking back and forth between the article and the edit page. He incorrectly copies the syntax of the link, although he recognises this to be some kind of source code with tags. He recognises his mistake and he tells me he would now check out the syntax by example, but it isn't really clear to him now.

**Changing population count** He finds the edit tab at the top again. First he changes the count in the sentence. While doing this, he notices the number in the template and changes this too, and saves the page.

### A.3.9 Participant 09

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
male	bachelor's student	often	never	block editor

**Fixing a typo** He clicks the edit tab at the top. He briefly reads the introduction. He then clicks the sentence, fixes the typo, and hits preview. He clicks preview without entering an edit summary.

**Adding a sentence** He clicks the tab at the top again, and scrolls to the sentence. He clicks the sentence at the end of the first paragraph, and adds a sentence.

I ask him what he thought about the experience so far, and he tells me that all the information at the top of the page is a bit unnecessary. We also talk a bit about the edit summary box. Overall though, he finds it to be pretty easy so far.

**Adding an external link** When on the edit page, he notices the different edit modes, and selects the section edit mode. He clicks the external link section and incorrectly copies the syntax of the existing links. He recognises his mistake and clicks the section again, quickly understanding the correct syntax and previewing again. Now this works, and he saves the page.

**Adding a reference** He clicks the section edit link at the references section, and is taken to the edit page of the entire page. He selects the section edit mode again and clicks the reference section. When previewing he notices a bug, and we refresh the page. He doesn't understand how the references work, and mistake the category links for references.

He sighs that he has no clue whatsoever how to accomplish this. He edits a sentence and just enters “[6]”. I tell him he can search Google, and he does this. He finds a page which doesn't clearly tell how this works. He thinks it might work like in L<sup>A</sup>T<sub>E</sub>X (which indeed is similar to how it's done here). He tries editing the references section again but finds that this still doesn't work.

He now tries to use the full editor. He scrolls through the article source code and heads to the references section again, with no avail. I ask him whether he would stop now, and he tells me he wouldn't stop. He still doesn't understand how the category links work. He heads back to the first editor and I suggest him to check out how the reference links in the text look like. He opens the full editor again and quickly understands the syntax. He copies another reference. We encounter a weird browser bug where the cursor disappears, but he manages to enter the reference anyway. He finds the save button which is inconsistent with the previous publish button.

**Changing population count** He uses the sentence edit mode to change the value in the text, which he then publishes. He doesn't seem to understand how to edit the info box, and defers to the full editor, where he quickly finds the template syntax and changes the value.

### A.3.10 Participant 10

Gender	Education	Wikipedia viewing	Wikipedia editing	Interface
male	bachelor's student	5 times a week	1 time	block editor

**Fixing a typo** He clicks the edit tab at the top of the page, and enters an edit summary by copying one of the examples below the text box. He looks at the instructions. He then clicks on the sentence and fixes the typo, hits preview, and clicks the publish button.

**Adding a sentence** He switches to the paragraph edit mode and clicks the first paragraph. He adds a sentence at the end of the paragraph, hits preview, enters an edit summary and hits publish.

**Adding an external link** He clicks the section edit link, and expects to only see the external links section, but instead gets the entire page again. He enters an edit summary. He notices he cannot edit the links in sentence edit mode, switches to section edit mode, and copies the syntax. He hits preview and then publishes the page.

**Adding a reference** Again, he clicks on the edit tab. He switches to section edit mode and clicks the reference section. He tries to copy the syntax of the category links. He hits preview and we encounter a bug in the software. I tell him he's on the wrong track trying to copy the category links.

He now tries to edit a reference in the text, and when looking at another reference he understands how this works. He incorrectly copies the syntax, but he copies it within a link. He edits the section again and asks me something about the syntax, to which I don't respond. He figures it out himself and it seems to work. I ask him how the syntax works, to which he correctly replies that the brackets generate links. He notices that the reference is still incorrect as there is no link description, but tells me he doesn't know how to fix this, enters an edit summary and publishes the page. He still doesn't really understand how references work.

**Changing population count** First, he uses the sentence edit mode to change the value in the text. He switches to section edit mode to change the value in the template, which works without problems. He enters an edit summary and hits publish.

# Appendix B

## Source code

This chapter contains a part of the source code of the MediaWiki extension.<sup>1</sup> The MediaWiki-specific code and stylesheets have been omitted as they are not relevant for the actual generation of the interface. Some Javascript code for the client is also included. Some things are specific to the hybrid interface, but the functional and block interfaces can also be generated with the same code, only different client side functionality has to be added for that.

Most of the basic techniques of Chapter 5 are implemented, such as partial parsing, collapsing of elements, and basic dependency detection. However, detection of misnesting is not included in this implementation. Also, as HTML entities are already immutable in the MediaWiki parser, these are used directly, instead of converting to HTML tags in a post processing stage.

The detection of elements is deliberately omitted in this listing, as this is not the focus of this thesis and is therefore very basic. Also, the handling of dependencies can only be done by forcing to reparse the entire page. Greying out the specific dependencies is not implemented.

### B.1 InlineEditorText.class.php

```
1 <?php
2 /**
3  * This class handles all the parsing of markings and wikitext to HTML.
4  *
5  * The process is divided in a few basic steps:
6  * - when doing a preview, the wikitext is changed and markings are shifted
7  * - have extensions add markings to this object based on the wikitext
8  * - when doing a preview, markings are matched up with previous markings
9  * - the markings are arranged in a tree structure
10 * - the node to be rendered is determined (initially, this would be the root)
11 * - when doing a preview, some markings are highlighted
12 * - the node to be rendered is being rendered to marked wikitext, and then parsed
13 * - when any extension objects to partial rendering (because of dependencies elsewhere
14 *   in the text), the entire page is re-rendered
15 */
16 class InlineEditorText implements Serializable {
17     private $wikiOriginal;    /// < original wikitext of the article
18     private $article;        /// < article object for parsing
19     private $markings;       /// < array of InlineEditorMarking objects
20     private $previous;       /// < array of InlineEditorMarking objects before this edit
21     private $editedPiece;    /// < InlineEditorMarking object to describe the range where ←
22         edits occurred
23     private $changedNode;    /// < Node which should be rendered when doing a partial ←
24         rendering
25     private $root;          /// < Root of the tree where the markings are arranged in
26     private $disableMarking; /// < Disable the marking process entirely, e.g. for dealing ←
27         with complex pages
28 }
29 /**
```

<sup>1</sup>The full implementation can be found online at <http://www.mediawiki.org/wiki/Extension:InlineEditor>.



```

27  * @param $article Article The article to work with.
28  */
29  public function __construct( Article $article ) {
30      $this->article = $article;
31      $this->wikiOriginal = '';
32      $this->disableMarking = false;
33  }
34
35  /**
36   * Load the original wikitext from $wikiText.
37   * @param $wikiText string
38   */
39  public function loadFromWikiText( $wikiText ) {
40      $this->wikiOriginal = $wikiText;
41  }
42
43  /**
44   * Get the original wikitext.
45   * @return string Original wikitext
46   */
47  public function getWikiOriginal() {
48      return $this->wikiOriginal;
49  }
50
51  /**
52   * Set or unset the disabling of marking
53   */
54  public function setDisableMarking( $value ) {
55      $this->disableMarking = $value;
56  }
57
58  /**
59   * Try to get a partial rendering of the page based on which part of the page has been ←
60     edited.
61   * Before and after partial rendering hooks are called (InlineEditorPartialBeforeParse and
62     InlineEditorPartialAfterParse), to have the ability to terminate in the case something ←
63     in
64     the partial rendering has a dependency elsewhere on the page. In this case, the entire ←
65     page
66     is re-rendered.
67   * @return array An array with the id of the object to replace, and the html to replace it ←
68     with
69   */
70  public function getPartialParserOutput() {
71      $this->process();
72
73      if( isset( $this->previous ) && $this->changedNode != $this->root ) {
74          $markedWiki = $this->changedNode->render();
75          if( wfRunHooks( 'InlineEditorPartialBeforeParse', array( &$markedWiki ) ) ) {
76              $output = $this->parse( $markedWiki );
77              if( wfRunHooks( 'InlineEditorPartialAfterParse', array( &$output ) ) ) {
78                  return array( 'id' => $this->changedNode->getId(), 'html' => $output->getText() );
79              }
80          }
81      }
82
83      return array( 'id' => $this->root->getId(), 'html' => $this->parseRoot()->getText() );
84  }
85
86  /**
87   * Get the full parser output.
88   * Use this method for initially rendering the page, and use
89   * getPartialParserOutput() for subsequent renderings.
90   * @return ParserOutput
91   */
92  public function getFullParserOutput() {
93      $this->process();
94      return $this->parseRoot();
95  }
96
97  /**
98   * Get an array which maps ids to original wikitext.
99   * @return array
100  */
101  public function getTexts() {
102      $this->process();
103
104      foreach( $this->markings as $id => $marking ) {
105          $texts[$id] = substr( $this->wikiOriginal, $marking->getStart(), $marking->getLength() )←
106      };

```

```

105     // force an empty string, as substr returns 'false'
106     if( !$texts[$id] ) $texts[$id] = '';
107 }
108
109     return $texts;
110 }
111
112
113 /**
114  * Add a marking to the list of markings. To be called by the different edit extensions.
115  * @param $marking InlineEditorMarking
116  */
117 public function addMarking( InlineEditorMarking $marking ) {
118     if ( !$marking->isValid() ) return;
119     $this->markings[$marking->getId() ] = $marking;
120 }
121
122 /**
123  * Handle an edit by replacing the wikitext, and shifting all markings on and after it.
124  * @param $id string Id handle of the edit
125  * @param $text string Text to change to
126  */
127 public function doEdit( $id, $text ) {
128     // abort if the marking is not known, else use $editMarking for convenience
129     if( !isset( $this->markings[$id] ) ) return;
130     $editMarking = $this->markings[$id];
131
132     $start = $editMarking->getStart();
133     $end = $editMarking->getEnd();
134
135     // calculate the offset; the difference in length between the original text and new text
136     $offset = strlen( $text ) - $editMarking->getLength();
137
138     // replace the original text by the new text
139     $this->wikiOriginal = substr( $this->wikiOriginal, 0, $start ) . $text . substr( $this->←
wikiOriginal, $end );
140
141     // only shift markings when the length actually changed
142     if( $offset != 0 ) {
143         foreach( $this->markings as $id => $marking ) {
144             // if the marking is strictly after the edited marking, shift both start and end ←
positions
145             if( $marking->getStart() > $end ) {
146                 $marking->setStart( $marking->getStart() + $offset );
147                 $marking->setEnd( $marking->getEnd() + $offset );
148             }
149             // if the marking is contained in the edited marking, only move the end position
150             elseif( $marking->getEnd() >= $start ) {
151                 $marking->setEnd( $marking->getEnd() + $offset );
152             }
153         }
154         // if the marking isn't valid anymore (length <= 0), remove it
155         if( !$marking->isValid() ) unset( $this->markings[$id] );
156     }
157 }
158
159 // add 'edited' and 'lastEdit' classes to the edited marking
160 // 'edited' will stay (yellow highlight), 'lastEdit' will be removed quickly
161 $editMarking->addClasses( array( 'edited', 'lastEdit' ) );
162
163 // store a copy of the edited marking to denote the range of changed wikitext
164 // this range might grow when trying to match new markings against these previous ←
markings
165 $this->editedPiece = clone $editMarking;
166
167 // store the markings as previous markings, and unset $this->markings
168 $this->previous = $this->markings;
169 unset( $this->markings );
170
171 // remove all 'lastEdit' classes so that when copying previous markings to new markings,
172 // no leftover classes are there
173 foreach( $this->previous as $marking ) {
174     $marking->removeClass( 'lastEdit' );
175 }
176 }
177
178 /**
179  * Give special treatment to parsing the root. Add the root divs only after parsing
180  * to make sure they survive the parsing.
181  */
182 protected function parseRoot() {
183     $output = $this->parse( $this->root->renderInside() );

```

```

184     $output->setText( $this->root->renderStartTag() . $output->getText() . $this->root->←
185         renderEndTag() );
186     return $output;
187 }
188 /**
189  * Have the wikitext marked by different extensions by calling the 'InlineEditorMark' hook←
190  * After that, tries to match previous markings against the new markings, and tries to ←
191     preserve
192  * the previous markings, while growing $this->editedPiece if needed.
193  */
194 protected function mark() {
195     // abort if we already did markings
196     if( isset( $this->markings ) ) return;
197     // initialise markings array
198     $this->markings = array();
199
200     // if marking is disabled, we want to terminate here
201     if( $this->disableMarking ) return;
202
203     // have the extensions mark the wikitext
204     wfRunHooks( 'InlineEditorMark', array( &$this ) );
205
206     // sort the markings while preserving the keys (ids)
207     uasort( $this->markings, 'InlineEditorText::sortByStartLengthLevel' );
208
209     // collapse markings
210     $this->collapseMarkings();
211
212     // match up previous markings
213     $this->matchPreviousMarkings();
214 }
215
216 /**
217  * Do all the main stuff: have markings (if needed), create the tree,
218  * find the changed node for the partial rendering, and apply highlights.
219  */
220 protected function process() {
221     if( isset( $this->root ) ) return;
222
223     $this->mark();
224
225     // add the root marking to the list after building the tree,
226     // so it will get in the list of markings, but isn't duplicated in the
227     // tree
228     $rootMarking = new InlineEditorRoot( $this->wikiOriginal );
229     $this->root = $this->buildTree( $this->markings, $rootMarking );
230     $this->markings[ $rootMarking->getId() ] = $rootMarking;
231
232     $this->changedNode = $this->findChangedNode();
233     $this->applyLastEditHighlight();
234 }
235
236 /**
237  * Parse the marked wikitext using $this->article.
238  * @param $wikiMarked string Marked wikitext
239  * @return ParserOutput
240  */
241 protected function parse( $wikiMarked ) {
242     global $wgParser;
243
244     // get the same parser options as usual, but remove [edit] links
245     $parserOptions = clone $this->article->getParserOptions();
246     $parserOptions->setEditSection( false );
247
248     // always remove the table of contents as we don't want it pop up at partial parsing
249     // or even at the first page render as it is an dependency
250     $wikiMarked .= "\n__NOTOC__";
251
252     // run $wikiMarked through the parser and store the result
253     return $wgParser->parse( $wikiMarked, $this->article->getTitle(),
254         $parserOptions, true, true, $this->article->getRevIdFetched() );
255 }
256
257 /**
258  * Finds markings of exact same positions, and uses only the deepest markings.
259  */
260 protected function collapseMarkings() {
261     foreach( $this->markings as $id => $marking ) {
262         if( isset( $previous ) ) {
263             if( $marking->getCollapsible() && $marking->samePositionAs( $previous ) ) {

```

```

264         unset( $this->markings[$previousID] );
265     }
266 }
267 $previous = $marking;
268 $previousID = $id;
269 }
270 }
271
272 /**
273  * Previous markings are moved into the current markings list to be able to only
274  * render a part of the page which is much faster.
275  *
276  * Match the previous markings by linearly running through two sorted lists;
277  * the current markings and the previous markings. Move one of the lists forward
278  * depending on the sorting conditions, which are start position (asc), length (desc)
279  * and class names (asc).
280  * Whenever a match is found, the previous marking is used to preserve the ids already
281  * present at the client so we don't have to re-render this piece. Whenever a mismatch
282  * occurs, $this->editedPiece grows to include the mismatch, because it needs to be
283  * re-rendered.
284  */
285 protected function matchPreviousMarkings() {
286     // abort if there is nothing to match
287     if( !isset( $this->previous ) ) return;
288
289     // don't use the root of the previous markings
290     unset( $this->previous['inline-editor-root'] );
291
292     // sort the previous markings, while *re-keying* to natural numbers (0, 1, 2, ...)
293     // this is necessary to be able to run through the array using an integer pointer
294     usort( $this->previous, 'InlineEditorText::sortByStartLengthLevel' );
295
296     // point to the start of the previous markings list
297     $indexPrevious = 0;
298     $newMarkings = array();
299     foreach( $this->markings as $marking ) {
300         // no match found yet
301         $foundMatch = false;
302
303         // walk through the list of previous markings until the end is reached, we're past the ←
304         // or it has been matched with the current marking
305         while( isset( $this->previous[$indexPrevious] ) ) {
306             $previous = $this->previous[$indexPrevious];
307
308             switch( self::sortByStartLengthLevel( $previous, $marking ) ) {
309                 case 1:
310                     // if we've moved past the current marking, break, mismatch, and go to the next ←
311                     // current marking
312                     break(2);
313                 case -1:
314                     // if we haven't moved past the current marking but also haven't found it, ←
315                     // continue the search
316                     $indexPrevious++;
317                     break;
318                 default:
319                     // a previous marking has been matched with a current marking
320                     // the previous marking will replace the current one
321                     $previous->setMatched( true );
322                     $newMarkings[$previous->getId()] = $previous;
323                     $foundMatch = true;
324                     $indexPrevious++;
325                     break(2);
326             }
327         }
328     }
329
330     if( !$foundMatch ) {
331         // a mismatch occurred, so use the new marking and have $this->editedPiece grow to ←
332         // include the new marking
333         $marking->setMatched( false );
334         $newMarkings[$marking->getId()] = $marking;
335         if( $marking->getStart() < $this->editedPiece->getStart() ) $this->editedPiece->←
336             setStart( $marking->getStart() );
337         if( $marking->getEnd() > $this->editedPiece->getEnd() ) $this->editedPiece->←
338             setEnd( $marking->getEnd() );
339     }
340
341     // replace $this->markings with the newly generated list and destroy the previous ←
342     // markings
343     $this->markings = $newMarkings;
344     unset( $this->previous );
345 }

```

```

340
341 /**
342 * Build a tree from an array of sorted (!) markings.
343 *
344 * @param $markingsSorted array A sorted array of InlineEditorMarking objects.
345 * @param $rootMarking InlineEditorRoot A root marking, not included in the list of sorted←
markings
346 * @return InlineEditorNode
347 */
348 protected function buildTree( array $markingsSorted, InlineEditorRoot $rootMarking ) {
349 // create the root
350 $root = new InlineEditorNode( $this->wikiOriginal, $rootMarking );
351
352 // $workingNode is the node we're trying to add children to
353 // initialise it to the root node
354 $workingNode = $root;
355
356 foreach( $markingsSorted as $marking ) {
357 // create a new node for this marking
358 $node = new InlineEditorNode( $this->wikiOriginal, $marking );
359
360 // keep trying to add $node to $workingNode, move a level up if it fails
361 while( true )
362 {
363 // try to add the node as a child to $workingNode
364 if( $workingNode->addChild( $node ) ) {
365 $workingNode = $node;
366 break;
367 }
368 else {
369 // if it doesn't work, try the parent of $workingNode
370 if( $workingNode == $root ) {
371 // if we're at the root, stop
372 // this should *never happen*!
373 break;
374 }
375 else {
376 $workingNode = $workingNode->getParent();
377 }
378 }
379 }
380 }
381 return $root;
382 }
383
384 /**
385 * Find the best (deepest) node that contains the edited text.
386 * @return InlineEditorPiece
387 */
388 protected function findChangedNode() {
389 // if there is no edited piece, just return the root
390 if( !isset( $this->editedPiece ) ) return $this->root;
391
392 // find the best (deepest) node that contains the edited text
393 $changedNode = $this->root->findBestParent( $this->editedPiece );
394
395 // go up as long as we're dealing with a node that hasn't been matched, which
396 // can only occur when $this->editedPiece coincides with multiple unmatched markings
397 // at exactly the same position - in that case the innermost node will be matched
398 // whereas we'd like to return the outermost node
399 while( $changedNode != $this->root && !$changedNode->getMarking()->getMatched() ) {
400 $changedNode = $changedNode->getParent();
401 }
402 return $changedNode;
403 }
404
405 /**
406 * Add highlights to all (highest) markings contained in the edited text.
407 */
408 protected function applyLastEditHighlight() {
409 // abort if there is no edited text
410 if( !isset( $this->editedPiece ) ) return;
411
412 // find the markings contained in $this->editedPiece and mark them
413 $children = $this->root->findBestChildren( $this->editedPiece );
414 foreach( $children as $child ) {
415 $child->getMarking()->addClasses( 'lastEdit edited' );
416 }
417 }
418
419 /**
420 * Sort function which sorts markings - in this particular order - on:
421 * - start position (asc)

```

```

422 * - length (desc)
423 * - level (desc)
424 * @param $a InlineEditorMarking
425 * @param $b InlineEditorMarking
426 * @return int
427 */
428 private static function sortByStartLengthLevel( $a, $b ) {
429     if( $a->getStart() == $b->getStart() ) {
430         if( $a->getLength() == $b->getLength() ) {
431             if( $a->getLevel() == $b->getLevel() ) {
432                 return $a->equals( $b, array( 'edited', 'lastEdit' ) ) ? 0 : 1;
433             }
434             else {
435                 return ( $a->getLevel() > $b->getLevel() ? -1 : 1 );
436             }
437         }
438         else {
439             return ( $a->getLength() > $b->getLength() ? -1 : 1 );
440         }
441     }
442     else {
443         return ( $a->getStart() < $b->getStart() ? -1 : 1 );
444     }
445 }
446
447 /**
448 * Serialize by doing a normal serialization of the original wikitext, the markings
449 * and the unique identifier to guarantee unique ids across the session. The serialization
450 * is base64 encoded to make sure it won't be screwed up by javascript.
451 */
452 public function serialize() {
453     return base64_encode( serialize( array(
454         'disableMarking' => $this->disableMarking,
455         'wikiOriginal'   => $this->wikiOriginal,
456         'markings'       => $this->markings,
457         'uniqueIdState'  => InlineEditorMarking::getUniqueIdState()
458     ) ) );
459 }
460
461 /**
462 * Unserialize, similar to serializing.
463 */
464 public function unserialize( $string ) {
465     $data = unserialize( base64_decode( $string ) );
466     $this->disableMarking = $data[ 'disableMarking' ];
467     $this->wikiOriginal   = $data[ 'wikiOriginal' ];
468     $this->markings       = $data[ 'markings' ];
469     InlineEditorMarking::setUniqueIdState( $data[ 'uniqueIdState' ] );
470 }
471
472 /**
473 * Initial state usable for the editor or API.
474 * @param $text InlineEditorText
475 * @return array Array containing the serialized object and an array of texts
476 */
477 public static function initialState( InlineEditorText $text ) {
478     return array( 'texts' => $text->getTexts(), 'object' => self::toSession( $text ) );
479 }
480
481 /**
482 * Subsequent state usable for the editor or API.
483 * @param $text InlineEditorText
484 * @return array Array containing the serialized object, an array of texts, and
485 * an array describing what html should be replaced
486 */
487 public static function subsequentState( InlineEditorText $text ) {
488     return array(
489         'texts' => $text->getTexts(),
490         'partialHtml' => $text->getPartialParserOutput(),
491         'object' => self::toSession( $text )
492     );
493 }
494
495 /**
496 * Restore the InlineEditorText object from a request array and article.
497 * @param $request array
498 * @param $article Article
499 * @return InlineEditorText
500 */
501 public static function restoreObject( array $request, Article $article ) {
502     $text = self::fromSession( $request[ 'object' ] );
503     $text->article = $article;
504     return $text;

```

```

505 }
506
507 /**
508  * Store the actual object in the session, as it can be quite large.
509  * @param $text InlineEditorText
510  * @return int Object identifier
511  */
512 protected static function toSession( $text ) {
513     $objectID = (isset($_SESSION['inline-editor-id']) ? $_SESSION['inline-editor-id'] + 1 : ←
514         0);
515     $_SESSION['inline-editor-id'] = $objectID;
516     $_SESSION['inline-editor-object-' . $objectID] = serialize( $text );
517     return $objectID;
518 }
519 /**
520  * Retrieve the object from the session.
521  * @param $object int Object identifier
522  * @return InlineEditorText
523  */
524 protected static function fromSession( $object ) {
525     return unserialize( $_SESSION['inline-editor-object-' . $object] );
526 }
527 }

```

## B.2 InlineEditorPiece.class.php

```

1 <?php
2 /**
3  * InlineEditorPiece is a base (abstract) class for everything that describes a start
4  * and end point, and provides some basic functionality. Subclasses have to implement
5  * getStart() and getEnd(), and then things like getLength(), equals(), etc. are provided
6  * for your convenience.
7  */
8 abstract class InlineEditorPiece {
9     /**
10     * Get the start position of the piece, to be implemented by an extending class.
11     * @return int
12     */
13     abstract public function getStart();
14
15     /**
16     * Get the end position of the piece, to be implemented by an extending class.
17     * @return int
18     */
19     abstract public function getEnd();
20
21     /**
22     * Get the length by taking the difference between getEnd() and getStart()
23     * @return int
24     */
25     public final function getLength() {
26         return $this->getEnd() - $this->getStart();
27     }
28
29     /**
30     * Check whether another piece has the exact same position as this one.
31     * @param $piece
32     * @return bool
33     */
34     public final function samePositionAs( InlineEditorPiece $piece ) {
35         return ( $piece->getStart() == $this->getStart() && $piece->getEnd() == $this->getEnd() ←
36             );
37     }
38
39     /**
40     * Check whether another piece is exactly the same as this one. By default this is
41     * the same as samePositionAs(), but it can be overridden by a subclass.
42     * @param $piece InlineEditorPiece
43     * @return bool
44     */
45     public function equals( InlineEditorPiece $piece ) {
46         return $this->samePositionAs( $piece );
47     }
48
49     /**
50     * Check whether another piece may fit inside this piece.

```

```

50  * @param $piece InlineEditorPiece
51  * @return bool
52  */
53  public final function canContain( InlineEditorPiece $piece ) {
54      return ( $piece->getStart() >= $this->getStart() &&
55              $piece->getStart() <= $this->getEnd() &&
56              $piece->getEnd() >= $this->getStart() &&
57              $piece->getEnd() <= $this->getEnd() );
58  }
59
60  /**
61  * Check whether another piece overlaps with this piece with one character or more.
62  * @param $piece InlineEditorPiece
63  * @return bool
64  */
65  public final function hasOverlap( InlineEditorPiece $piece ) {
66      return ( $piece->getStart() < $this->getEnd() && $piece->getEnd() > $this->getStart() );
67  }
68
69  /**
70  * Check whether another piece touches this piece at the start or end.
71  * @param $piece InlineEditorPiece
72  * @return bool
73  */
74  public final function touches( InlineEditorPiece $piece ) {
75      return ( $piece->getStart() == $this->getEnd() || $piece->getEnd() == $this->getStart() ) <-
76  };
77  }
78
79  /**
80  * Simple check to prevent invalid values.
81  * @return bool
82  */
83  public function isValid() {
84      return $this->getEnd() > $this->getStart();
85  }

```

### B.3 InlineEditorMarking.class.php

```

1  <?php
2  /**
3   * This class provides a way of setting markings that apply to wikitext, and store some <-
4   * basic information
5   * on them, like a list of classnames, an id which is automatically generated, whether or <-
6   * not it is inline,
7   * and whether or not it has been matched before.
8   */
9  class InlineEditorMarking extends InlineEditorPiece {
10     const defaultClasses = 'inlineEditorElement notEditing'; // default classes; unfortunately <-
11     we cannot use an array here
12     const autoClasses = 'block inline bar nobar'; // automatically added classes which shouldn't <-
13     be added by hand
14     protected static $lastId = 0; /// < counter which is used to generate unique ids
15
16     protected $start;          /// < start position of the marking in the wikitext
17     protected $end;            /// < end position of the marking in the wikitext
18     protected $classes;        /// < class(es) attached to the marking which identifies the type
19     protected $block;          /// < whether the tag should be added as a block or inline
20     protected $bar;            /// < whether the text should carry a bar at the left, or be fully <-
21     selectable
22     protected $level;          /// < nesting level, which is used to sort consistently when two <-
23     markings are of same length
24     protected $collapsible;     /// < whether or not the marking can be collapsed by removing <-
25     parent elements of the same size
26
27     protected $id;              /// < id in the original text; this will be unique even when <-
28     calculating new ids!
29     protected $matched;         /// < bool whether or not this marking has been matched with a <-
30     previous marking (default: true)
31
32     /**
33     * @param $start      int Start of the marking, offset in number of characters from the <-
34     * begin of the wikitext
35     * @param $end        int End of the marking, offset in number of characters from the <-
36     * begin of the wikitext

```



```

26 * @param $classes    mixed Class(es) the marking should be labeled with, can be either a↔
    string or an array of strings
27 * @param $block     bool Whether the tag should be added as a block or inline
28 * @param $bar       bool Whether the text should carry a bar at the left, or be fully ↔
    selectable
29 * @param $level     int Nesting level, which is used to sort consistently when two ↔
    markings are of same length, default: 0
30 * @param $collapsible bool Whether or not the marking can be collapsed by removing parent↔
    elements of the same size
31 */
32 function __construct( $start, $end, $classes, $block, $bar, $level = 0, $collapsible = ↔
    true ) {
33     $this->start      = $start;
34     $this->end        = $end;
35     $this->block      = $block;
36     $this->bar        = $bar;
37     $this->level      = $level;
38     $this->collapsible = $collapsible;
39
40     $this->matched    = true;
41     $this->id         = self::uniqueId();
42
43     $this->classes     = array();
44     $this->addClasses( $classes );
45 }
46
47 /**
48 * Get the start position in the original wikitext.
49 * @return int
50 */
51 public function getStart() {
52     return $this->start;
53 }
54
55 /**
56 * Get the end position in the original wikitext.
57 * @return int
58 */
59 public function getEnd() {
60     return $this->end;
61 }
62
63 /**
64 * Move the start position. The caller should check the validity using isValid().
65 * @param $pos int
66 */
67 public function setStart( $pos ) {
68     $this->start = $pos;
69 }
70
71 /**
72 * Move the end position. The caller should check the validity using isValid().
73 * @param $pos int
74 */
75 public function setEnd( $pos ) {
76     $this->end = $pos;
77 }
78
79 /**
80 * Get an array of classes that have been assigned to this marking.
81 * @return array
82 */
83 public function getClasses() {
84     return $this->classes;
85 }
86
87 /**
88 * Get a space separated list of the classes that have been assigned to this marking.
89 * @return string
90 */
91 public function getClass() {
92     return implode( ' ', $this->classes );
93 }
94
95 /**
96 * Check whether or not a certain class has been assigned to this marking.
97 * @param $class string
98 * @return bool
99 */
100 public function hasClass( $class ) {
101     return in_array( $class, $this->classes );
102 }
103

```

```

104  /**
105  * Get the unique ID of this marking.
106  * @return string
107  */
108  public function getId() {
109      return $this->id;
110  }
111
112  /**
113  * Add a class to the list of assigned classes.
114  * @param $class string
115  */
116  public function addClass( $class ) {
117      $this->addClasses( $class );
118  }
119
120  /**
121  * Add classes to the list of assigned classes.
122  * @param $classes mixed Either a space separated string or an array.
123  */
124  public function addClasses( $classes ) {
125      // convert space separated to an array
126      if( !is_array($classes) ) $classes = explode( ' ', $classes );
127
128      // exclude the default classes that are always included
129      $classes = array_diff( $classes, self::getDefaultClassesArray() );
130
131      // also exclude classes that are automatically included
132      $classes = array_diff( $classes, self::getAutoClassesArray() );
133
134      // merge with the current classes and remove duplicates
135      $this->classes = array_unique( array_merge( $this->classes, $classes ) );
136  }
137
138  /**
139  * Remove a class from the list of assigned classes.
140  * @param $class string
141  */
142  public function removeClass( $class ) {
143      $this->removeClasses( $class );
144  }
145
146  /**
147  * Remove classes from the list of assigned classes.
148  * @param $classes mixed Either a space separated string or an array.
149  */
150  public function removeClasses( $classes ) {
151      // convert space separated to an array
152      if( !is_array($classes) ) $classes = explode( ' ', $classes );
153
154      // exclude the default classes that are always included
155      $classes = array_diff( $classes, self::getDefaultClassesArray() );
156
157      // also exclude classes that are automatically included
158      $classes = array_diff( $classes, self::getAutoClassesArray() );
159
160      // save the difference between the existing classes and the classes we're removing
161      $this->classes = array_diff( $this->classes, $classes );
162  }
163
164  /**
165  * Get whether or not this marking has been matched with a previous marking.
166  * Default value is true.
167  * @return bool
168  */
169  public function getMatched() {
170      return $this->matched;
171  }
172
173  /**
174  * Set whether or not this marking has been matched with a previous marking.
175  * @param $value bool
176  */
177  public function setMatched( $value ) {
178      $this->matched = $value;
179  }
180
181  /**
182  * Get the nesting level, which is used to sort consistently when two markings are of same←
183  * length.
184  * @param $value int
185  */
186  public function getLevel() {

```

```

186     return $this->level;
187 }
188
189 /**
190  * Get whether or not the marking can be collapsed by removing parent elements of the same←
191     size
192     * @param $value int
193     */
194 public function getCollapsible() {
195     return $this->collapsible;
196 }
197
198 /**
199  * Overrides InlineEditorPiece to be able to check for equality between markings,
200  * where also classes are compared. This also adds the ability to ignore certain
201  * classes when comparing (such as classes for what has been edited, etc.)
202  * @param $piece InlineEditorPiece Piece or marking to compare with
203  * @param $ignoreClasses array Array of classes that should be ignored
204  * @return bool
205     */
206 public function equals( InlineEditorPiece $piece, array $ignoreClasses = array() ) {
207     if( $piece instanceof InlineEditorMarking ) {
208         $classesA = array_diff( $this->classes, $ignoreClasses );
209         $classesB = array_diff( $piece->classes, $ignoreClasses );
210         $diffA    = array_diff( $classesA, $classesB );
211         $diffB    = array_diff( $classesB, $classesA );
212         return ( parent::equals( $piece ) && empty( $diffA ) && empty( $diffB ) );
213     }
214     else {
215         return parent::equals( $piece );
216     }
217 }
218
219 /**
220  * Render the open tag with classes and id. Depending on $this->block there will be
221  * a newline after the tag, or not.
222  * @return string HTML
223     */
224 public function renderStartTag() {
225     $attrs = array( 'class' => $this->getFullClass(), 'id' => $this->id );
226     return HTML::openElement( 'div', $attrs ) . $this->getNewline();
227 }
228
229 /**
230  * Render the close tag (</div>) with an extra newline before it if $this->block.
231  * @return string HTML
232     */
233 public function renderEndTag() {
234     return $this->getNewline() . '</div>';
235 }
236
237 /**
238  * Get the full class string to render. Includes the default classes
239  * for more convenient CSS, and depending on the class values
240  * additionally 'block' or 'inline', plus 'bar' or 'nobar'.
241  * @return string Space separated classes
242     */
243 protected function getFullClass() {
244     return $this->getClass() . ' ' . self::defaultClasses
245         . ( $this->block ? ' block' : ' inline' )
246         . ( $this->bar ? ' bar' : ' nobar' );
247 }
248
249 /**
250  * Get a newline when $this->block is set or else an empty string.
251  * @return string Newline or nothing
252     */
253 protected function getNewline() {
254     return $this->block ? "\n": '';
255 }
256
257 /**
258  * Get an array version of the default classes that should always be included
259  * @return array
260     */
261 protected static function getDefaultClassesArray() {
262     return explode( ' ', self::defaultClasses );
263 }
264
265 /**
266  * Get an array version of the automatically added classes
267  * @return array
268     */

```

```

268     protected static function getAutoClassesArray() {
269         return explode( ' ', self::autoClasses );
270     }
271
272     /**
273      * Get a unique id by using self::$lastId and incrementing it.
274      * @return string
275      */
276     protected static function uniqueId() {
277         return 'inline-editor-' . self::$lastId++;
278     }
279
280     /**
281      * Returns the $lastId variable in order to preserve it across requests.
282      * @return int Store this integer and pass it to setUniqueIdState() later on
283      */
284     public static function getUniqueIdState() {
285         return self::$lastId;
286     }
287
288     /**
289      * Set the $lastId variable in order to preserve it across requests.
290      * @param $state int State acquired by getUniqueIdState()
291      */
292     public static function setUniqueIdState( $state ) {
293         if( $state > self::$lastId ) self::$lastId = $state;
294     }
295 }

```

## B.4 InlineEditorRoot.class.php

```

1 <?php
2 /**
3  * This is a special marking that spans all wikitext.
4  */
5 class InlineEditorRoot extends InlineEditorMarking {
6     function __construct( &$wiki ) {
7         parent::__construct( 0, strlen( $wiki ), 'rootElement inlineEditorBasic', true, true, 0, ←
8             false );
9         $this->id = 'inline-editor-root';
10 }

```

## B.5 InlineEditorNode.class.php

```

1 <?php
2 /**
3  * This class wraps an InlineEditorMarking to be a part of a tree spanning the
4  * wikitext. It is closely connected to the wikitext, and should be recreated whenever a
5  * marking or wikitext changes.
6  */
7 class InlineEditorNode extends InlineEditorPiece {
8     protected $wiki;      ///< reference to the original wikitext
9     protected $children;  ///< array of children (InlineEditorNode)
10    protected $isSorted;  ///< bool whether or not the children are sorted
11    protected $lastEnd;   ///< largest endposition of children, to verify during adding the ←
12                          children are sortd
13    protected $marking;   ///< marking this nodes wraps
14    protected $parent;    ///< parent node (InlineEditorNode)
15
16    /**
17     * @param $wiki String Reference to the original wikitext
18     * @param $marking InlineEditorMarking Marking to wrap in the tree
19     */
19    public function __construct( &$wiki, InlineEditorMarking $marking ) {
20        $this->wiki =& $wiki;
21        $this->children = array();
22        $this->lastEnd = 0;
23        $this->isSorted = true;
24        $this->marking = $marking;

```

```

25 }
26
27 /**
28  * Get the start position from the corresponding marking.
29  * @return int
30  */
31 public function getStart() {
32     return $this->marking->getStart();
33 }
34
35 /**
36  * Get the end position from the corresponding marking.
37  * @return int
38  */
39 public function getEnd() {
40     return $this->marking->getEnd();
41 }
42
43 /**
44  * Get the id from the corresponding marking.
45  * @return string
46  */
47 public function getId() {
48     return $this->marking->getId();
49 }
50
51 /**
52  * Get an array of children of type InlineEditorNode.
53  * @return array
54  */
55 public function getChildren() {
56     return $this->children;
57 }
58
59 /**
60  * Get the corresponding marking.
61  * @return InlineEditorMarking
62  */
63 public function getMarking() {
64     return $this->marking;
65 }
66
67 /**
68  * Get the parent node.
69  * @return InlineEditorNode
70  */
71 public function getParent() {
72     return $this->parent;
73 }
74
75 /**
76  * Render the start tag by calling the corresponding marking.
77  * @return string HTML
78  */
79 public function renderStartTag() {
80     return $this->marking->renderStartTag();
81 }
82
83 /**
84  * Render the end tag by calling the corresponding marking.
85  * @return string HTML
86  */
87 public function renderEndTag() {
88     return $this->marking->renderEndTag();
89 }
90
91 /**
92  * Add a node to the list of children.
93  *
94  * Checks whether or not the child can be added. Returns false if it cannot add the
95  * child, and true when it can. The calling class is responsible to add the node to
96  * the innermost node, this will not be done by the function.
97  *
98  * It is recommended to add nodes from left to right, as this gives the best performance.
99  *
100  * @param $child InlineEditorNode
101  * @return bool
102  */
103 public function addChild( InlineEditorNode $child ) {
104     // if we cannot contain the child, we cannot add it
105     if( !$this->canContain( $child ) ) return false;
106
107     // if the start is before the largest endpoint, check all children for overlap

```

```

108     if( $child->getStart() < $this->lastEnd ) {
109         foreach( $this->children as $otherChild ) {
110             if( $child->hasOverlap( $otherChild ) ) return false;
111         }
112         // if there is no overlap, we're sure that the list isn't sorted anymore
113         $this->isSorted = false;
114     }
115
116     // add the child and set the parent of the child to $this
117     $this->children[$child->getStart()] = $child;
118     $child->parent = $this;
119
120     // move $this->lastEnd if needed
121     if( $child->getEnd() > $this->lastEnd ) $this->lastEnd = $child->getEnd();
122
123     return true;
124 }
125
126 /**
127  * Find the node with the smallest length still able to contain $piece.
128  * @param $piece InlineEditorPiece
129  * @return InlineEditorNode
130  */
131
132 public function findBestParent( InlineEditorPiece $piece ) {
133     // if we cannot contain the piece, return false
134     if( !$this->canContain( $piece ) ) return false;
135
136     // sorted children is a precondition for the algorithm
137     $this->sort();
138
139     foreach( $this->children as $start => $child ) {
140         // if we've move past the end of the piece, stop
141         if( $piece->getEnd() < $start ) break;
142
143         // try to fit the piece to this child
144         if( $piece->getStart() >= $start ) {
145             $fit = $child->findBestParent( $piece );
146             // if we found a child that fits the piece, return it
147             if( $fit !== false ) {
148                 return $fit;
149             }
150         }
151     }
152
153     // if we cannot find a suitable child, but we can contain it in this piece, return $this
154     return $this;
155 }
156
157 /**
158  * Find the highest level of children that can be fit into a certain piece.
159  * This will return an array of nodes that are best fit.
160  * @param $piece InlineEditorPiece
161  * @return array
162  */
163
164 public function findBestChildren( InlineEditorPiece $piece ) {
165     // try to find a parent that fits $piece (which can very well be $this!)
166     $parent = $this->findBestParent( $piece );
167
168     // if we cannot find a parent, return false
169     if( !$parent ) return false;
170
171     // if the piece can contain the entire parent piece, just return that piece
172     if( $piece->canContain( $parent ) ) return array( $parent );
173
174     // sorting is a precondition of the algorithm
175     $this->sort();
176
177     $children = array();
178     foreach( $parent->children as $start => $child ) {
179         // if we've moved past the end of the piece, stop
180         if( $start > $piece->getEnd() ) break;
181
182         // add the child to the list if it can be contained in the piece
183         if( $piece->canContain( $child ) ) {
184             $children[] = $child;
185         }
186     }
187     return $children;
188 }
189
190 /**
191  * Render the entire tag, with recursion on the children.

```

```

191     * @return string HTML
192     */
193     public function render() {
194         return $this->renderStartTag() . $this->renderInside() . $this->renderEndTag();
195     }
196
197     /**
198     * Render the inside of the tag.
199     */
200     public function renderInside() {
201         $this->sort();
202         $lastPos = $this->getStart();
203         $output = '';
204         foreach( $this->children as $child ) {
205             $output .= substr( $this->wiki, $lastPos, $child->getStart() - $lastPos );
206             $output .= $child->render();
207             $lastPos = $child->getEnd();
208         }
209
210         $output .= substr( $this->wiki, $lastPos, $this->getEnd() - $lastPos );
211         return $output;
212     }
213
214     /**
215     * Sort the children by start position (key).
216     */
217     protected function sort() {
218         if( $this->isSorted ) return;
219         ksort( $this->children );
220         $this->isSorted = true;
221     }
222 }

```

## B.6 jquery.inlineEditor.js

```

1  /**
2  * Client side framework of the InlineEditor. Facilitates publishing, previewing,
3  * using specific editors, and undo/redo operations.
4  */
5  ( function( $ ) { $.inlineEditor = {
6      editors: {},
7
8      states: [], // history of all the states (HTML and original wikitexts)
9      currentState: 0, // state that is currently viewed
10     lastState: 0, // last state in the history
11     publishing: false, // whether or not currently publishing
12
13     /**
14     * Adds the initial state from the current HTML and a wiki string.
15     */
16     addInitialState: function( state ) {
17         $.inlineEditor.currentState = 0;
18         $.inlineEditor.states[$.inlineEditor.currentState] = {
19             'object': state.object,
20             'texts': state.texts,
21             'html': $( '#editContent' ).html()
22         };
23     },
24
25     /**
26     * Returns wikitext in the current state given an ID.
27     */
28     getTextById: function( id ) {
29         return $.inlineEditor.states[$.inlineEditor.currentState].texts[id];
30     },
31
32     /**
33     * Previews given a new text for a given field by ID.
34     */
35     previewTextById: function( text, id ) {
36         // send out an AJAX request which will be handled by addNewState()
37         var data = {
38             'object': $.inlineEditor.states[$.inlineEditor.currentState].object,
39             'lastEdit': { 'id': id, 'text': text }
40         };
41
42         var args = [ $.toJSON( data ), wgPageName ];

```

```

43     sajax_request_type = 'POST';
44     sajax_do_call( 'InlineEditor::ajaxPreview', args, $.inlineEditor.addNewState );
45 },
46
47 /**
48  * Adds a new state from an AJAX request.
49  */
50 addNewState: function( request ) {
51     var state = $.parseJSON( request.responseText );
52
53     // restore the html to the current state, instantly remove the lastEdit,
54     // and then add the new html
55     $( '#editContent' ).html( $.inlineEditor.states[ $.inlineEditor.currentState ].html );
56     $( '.lastEdit' ).removeClass( 'lastEdit' );
57     $( '#' + state.partialHtml.id ).replaceWith( state.partialHtml.html );
58
59     // add the new state
60     $.inlineEditor.currentState += 1;
61     $.inlineEditor.states[ $.inlineEditor.currentState ] = {
62         'object': state.object,
63         'texts': state.texts,
64         'html': $( '#editContent' ).html()
65     };
66
67     // clear out all states after the current state, because undo/redo would be broken
68     var i = $.inlineEditor.currentState + 1;
69     while( i <= $.inlineEditor.lastState ) {
70         delete $.inlineEditor.states[i];
71         i += 1;
72     }
73     $.inlineEditor.lastState = $.inlineEditor.currentState;
74
75     // reload the current editor and update the edit counter
76     $.inlineEditor.reload();
77     $.inlineEditor.updateEditCounter();
78 },
79
80 /**
81  * Cancels any open editor.
82  */
83 cancel: function() {
84     for( var optionNr in $.inlineEditor.editors ) {
85         $.inlineEditor.editors[optionNr].cancel();
86     }
87 },
88
89 /**
90  * Reloads the current editor and finish some things in the HTML.
91  */
92 reload: function() {
93     // cancel all editing
94     $.inlineEditor.cancel();
95
96     // reload the editors
97     for( var optionNr in $.inlineEditor.editors ) {
98         $.inlineEditor.editors[optionNr].reload();
99     }
100
101     // remove all lastEdit elements
102     $( '.lastEdit' ).removeClass( 'lastEdit' );
103
104     // make the links in the article unusable
105     $( '#editContent a' ).click( function( event ) { event.preventDefault(); } );
106 },
107
108 /**
109  * Moves back one state.
110  */
111 undo: function( event ) {
112     event.stopPropagation();
113     event.preventDefault();
114
115     // check if we can move backward one state and do it
116     if( $.inlineEditor.currentState > 0 ) {
117         $.inlineEditor.currentState -= 1;
118         $( '#editContent' ).html( $.inlineEditor.states[ $.inlineEditor.currentState ].html );
119         $.inlineEditor.reload();
120     }
121
122     // refresh the edit counter regardless of actually switching, this confirms
123     // that the button works, even if there is nothing to switch to
124     $.inlineEditor.updateEditCounter();
125 },

```



```

126
127 /**
128  * Moves forward one state.
129  */
130 redo: function( event ) {
131     event.stopPropagation();
132     event.preventDefault();
133
134     // check if we can move forward one state and do it
135     if( $.inlineEditor.currentState < $.inlineEditor.lastState ) {
136         $.inlineEditor.currentState += 1;
137         $('#editContent').html( $.inlineEditor.states[$.inlineEditor.currentState].html );
138         $.inlineEditor.reload();
139     }
140
141     // refresh the edit counter regardless of actually switching, this confirms
142     // that the button works, even if there is nothing to switch to
143     $.inlineEditor.updateEditCounter();
144 },
145
146 /**
147  * Updates the edit counter and makes it flash.
148  */
149 updateEditCounter: function() {
150     // update the value of the edit counter
151     var $editCounter = $( '#editCounter' );
152     $editCounter.text( '#' + $.inlineEditor.currentState );
153
154     // remove everything from the editcounter, and have it fade again
155     $editCounter.stop(true, true).hide().fadeIn('fast');
156 },
157
158 /**
159  * Show the interface for a particular element.
160  * @return Boolean Whether or not showing the interface was successful.
161  */
162 show: function( id ) {
163     // disable the existing editing field if necessary
164     $.inlineEditor.editors.basic.cancel();
165
166     // try the show function of all editors
167     for( var optionNr in $.inlineEditor.editors ) {
168         if( $.inlineEditor.editors[optionNr].show( id ) ) return true;
169     }
170     return false;
171 },
172
173 /**
174  * Submit event, adds the json to the hidden field
175  */
176 submit: function( event ) {
177     $.inlineEditor.publishing = true;
178
179     // get the wikitext from the state as it's currently on the screen
180     var data = {
181         'object': $.inlineEditor.states[$.inlineEditor.currentState].object
182     };
183     var json = $.toJSON( data );
184
185     // set and send the form
186     $( '#json' ).val( json );
187 },
188
189 /**
190  * Publishes the document
191  */
192 publish: function() {
193     $( '#editForm' ).submit();
194 },
195
196 warningMessage: function( ) {
197     if ( $.inlineEditor.lastState > 0 && !$.inlineEditor.publishing ) {
198         return mediaWiki.msg( 'vector-editwarning-warning' );
199     }
200 },
201
202 enableEditWarning: function( ) {
203     window.onbeforeunload = $.inlineEditor.warningMessage;
204 },
205
206 /**
207  * Initializes the editor.
208  */

```

```

209 init : function() {
210     $( '#editForm' ).submit( $.inlineEditor.submit );
211     $( '#publish' ).click( $.inlineEditor.publish );
212     mw.util.updateTooltipAccessKeys( $( '#publish' ) );
213
214     if( $( '#advancedbox' ).size() > 0 ) {
215         $( '#undo' ).click( $.inlineEditor.undo );
216         mw.util.updateTooltipAccessKeys( $( '#undo' ) );
217
218         $( '#redo' ).click( $.inlineEditor.redo );
219         mw.util.updateTooltipAccessKeys( $( '#redo' ) );
220     }
221
222     // reload the current editor
223     $.inlineEditor.reload();
224 }
225
226 }; } ) ( jQuery );

```

## B.7 jquery.inlineEditor.editors.basic.js

```

1 /**
2  * Provides a basic editor with preview and cancel functionality.
3  */
4 ( function( $ ) { $.inlineEditor.editors.basic = {
5
6     /**
7      * Creates a new hovering edit field.
8      */
9     newField: function( $field, originalClickEvent ) {
10         // create a new field
11         var $newField = $( '<' + $field.get(0).nodeName + '>' );
12         $newField.addClass( 'editing' );
13
14         // position the field floating on the page, at the same position the original field
15         $newField.css( 'top', $field.position().top );
16
17         // point to the original field using jQuery data
18         $newField.data( 'orig', $field );
19
20         // make sure click and mousemove events aren't passed on
21         $newField.click( function( event ) { event.stopPropagation(); } );
22         $newField.mousemove( function( event ) { event.stopPropagation(); } );
23
24         // add the field after the current field in code
25         $field.after( $newField );
26         return $newField;
27     },
28
29     /**
30      * Adds an edit bar to the field with preview and cancel functionality.
31      */
32     addEditBar: function( $newSpan, wiki ) {
33         // build the input field
34         var $input = $( '<textarea></textarea>' );
35         $input.text( wiki );
36
37         // build preview and cancel buttons and add click events
38         var $preview = $( '<input type="button" class="preview"/>' );
39         $preview.attr( 'value', mediaWiki.msg( 'inline-editor-preview' ) );
40         $preview.attr( 'accesskey', mediaWiki.msg( 'accesskey-inline-editor-preview' ) );
41         $preview.attr( 'title', mediaWiki.msg( 'tooltip-inline-editor-preview' ) +
42             ' | ' + mediaWiki.msg( 'accesskey-inline-editor-preview' ) + ' ]' );
43         $preview.click( $.inlineEditor.editors.basic.clickPreview );
44
45         var $cancel = $( '<input type="button" class="cancel"/>' );
46         $cancel.attr( 'value', mediaWiki.msg( 'inline-editor-cancel' ) );
47         $cancel.attr( 'accesskey', mediaWiki.msg( 'accesskey-inline-editor-cancel' ) );
48         $cancel.attr( 'title', mediaWiki.msg( 'tooltip-inline-editor-cancel' ) +
49             ' | ' + mediaWiki.msg( 'accesskey-inline-editor-cancel' ) + ' ]' );
50         $cancel.click( $.inlineEditor.editors.basic.clickCancel );
51
52         // fix access key tooltips
53         mw.util.updateTooltipAccessKeys( $preview );
54         mw.util.updateTooltipAccessKeys( $cancel );
55
56         // build a div for the buttons

```

```

57     var $buttons = $( '<div class="buttons"></div> ' );
58     $buttons.append( $preview );
59     $buttons.append( $cancel );
60
61     // build the edit bar from the input field and buttons
62     var $editBar = $( '<div class="editbar"></div>' );
63     $editBar.append( $input );
64     $editBar.append( $buttons );
65
66     // append the edit bar to the new span
67     $newSpan.append( $editBar );
68
69     // automatically resize the textarea using the Elastic plugin
70     $input.elastic();
71
72     // focus on the input so you can start typing immediately
73     $input.focus();
74
75     return $editBar;
76 },
77
78 /**
79  * Default click handler for simple editors.
80  */
81 click: function( event ) {
82     var $field = $(this);
83
84     if( $field.hasClass( 'nobar' ) || event.pageX - $field.offset().left < 10 ) {
85         // prevent clicks from reaching other elements
86         event.stopPropagation();
87         event.preventDefault();
88
89         // disable the existing editing field if necessary
90         $.inlineEditor.editors.basic.cancel();
91
92         $.inlineEditor.editors.basic.show( $field.attr( 'id' ) );
93     }
94 },
95
96 /**
97  * Actually handles showing the editing interface. Recommended to override.
98  * @return Boolean Whether or not showing the interface was successful.
99  */
100 show: function( id ) {
101     $field = $( '#' + id );
102
103     // if the class is incorrect, terminate
104     if( !$field.hasClass( 'inlineEditorBasic' ) ) return false;
105
106     // find the element and retrieve the corresponding wikitext
107     var wiki = $.inlineEditor.getTextById( $field.attr( 'id' ) );
108
109     // create the edit field and build the edit bar
110     var $newField = $.inlineEditor.editors.basic.newField( $field, $.inlineEditor.editors.↵
        basic.click );
111     $.inlineEditor.editors.basic.addEditBar( $newField, wiki );
112
113     // add the wikiEditor toolbar
114     if( $.fn.wikiEditor ) {
115         $textarea = $newField.find( 'textarea' );
116
117         if( $.wikiEditor.modules.toolbar && $.wikiEditor.modules.toolbar.config && $.↵
            wikiEditor.isSupported( $.wikiEditor.modules.toolbar ) ) {
118             $textarea.wikiEditor( 'addModule', $.wikiEditor.modules.toolbar.config.↵
                getDefaultConfig() );
119         }
120
121         if( $.wikiEditor.modules.dialogs && $.wikiEditor.modules.dialogs.config && $.↵
            wikiEditor.isSupported( $.wikiEditor.modules.dialogs ) ) {
122             $.wikiEditor.modules.dialogs.config.replaceIcons( $textarea );
123             $textarea.wikiEditor( 'addModule', $.wikiEditor.modules.dialogs.config.↵
                getDefaultConfig() );
124         }
125     }
126     return true;
127 },
128
129 /**
130  * Cancels the current edit operation.
131  */
132 clickCancel: function( event ) {
133     // prevent clicks from reaching other elements
134     event.stopPropagation();

```

```

135     event.preventDefault();
136
137     // find the outer span, three parents above the buttons
138     var $span = $(this).parent().parent().parent();
139
140     // find the span with the original value
141     var $orig = $span.data( 'orig' );
142
143     // convert the span to it's original state
144     $orig.removeClass( 'orig' );
145     $orig.removeClass( 'hover' );
146
147     // place the original span after the current span and remove the current span
148     $span.after( $orig );
149     $span.remove();
150
151     // reload the editor to fix stuff that might or might not be broken
152     $.inlineEditor.reload();
153 },
154
155 /**
156  * Previews the current edit operation.
157  */
158 clickPreview: function( event ) {
159     // prevent clicks from reaching other elements
160     event.stopPropagation();
161     event.preventDefault();
162
163     // find the span with class 'editbar'
164     var $editbar = $(this).closest( '.editbar' );
165
166     // the element is one level above the editbar
167     var $element = $editbar.parent();
168
169     // add a visual indicator to show the preview is loading
170     $element.addClass( 'saving' );
171     var $overlay = $( '<div class="overlay"><div class="alpha"></div></div>' );
173     $editbar.after( $overlay );
174
175     // get the edited text and the id to save it to
176     var text = $editbar.find( 'textarea' ).val();
177     var id = $element.data( 'orig' ).attr( 'id' );
178
179     // let the inlineEditor framework handle the preview
180     $.inlineEditor.previewTextById( text, id );
181 },
182
183 /**
184  * Reload the editor.
185  */
186 reload: function() {
187     $.inlineEditor.editors.basic.bindEvents( $( '.inlineEditorBasic' ) );
188 },
189
190 /**
191  * Cancel all basic editors.
192  */
193 cancel: function() {
194     $( '.editing' ).find( '.cancel' ).click();
195 },
196
197 /**
198  * Bind all required events.
199  */
200 bindEvents: function( $elements ) {
201     $elements.unbind();
202     $elements.click( $.inlineEditor.editors.basic.click );
203     $elements.mousemove( $.inlineEditor.editors.basic.mouseMove );
204     $elements.mouseleave( $.inlineEditor.editors.basic.mouseLeave );
205 },
206
207 /**
208  * Do a javascript hover on the bars at the left.
209  */
210 mouseMove: function( event ) {
211     var $field = $( this );
212     if( $field.hasClass( 'bar' ) ) {
213         if( event.pageX - $field.offset().left < 10 ) {
214             $field.addClass( 'hover' );
215         }
216     }
217     else {
218         $field.removeClass( 'hover' );
219     }
220 }

```

```
217     }
218   }
219 },
220
221 /**
222  * Remove the hover class when leaving the element.
223  */
224 mouseleave: function( event ) {
225   var $field = $( this );
226   if( $field.hasClass( 'bar' ) ) {
227     $field.removeClass( 'hover' );
228   }
229 }
230
231 }; } ) ( jQuery );
```

# Appendix C

## Initial prototypes

The first thing shown to the MediaWiki community were some prototypes.<sup>1</sup> Screenshots of these initial prototypes are shown here. We also included the original motivations for choices made, which are important, as these choices are still visible in the in-line editing interfaces presented in this thesis.

### C.1 Prototype 1

The first prototype is simply a proof of concept (Figure C.1). It shows the basic idea of splitting an article into sentences, which can be edited, one at a time. The prototype is not actually for use on a live website, but it does show the basic usability ideas behind it.

First of all, the sentences that are editable are highlighted with a light blue colour. This colour has primarily been chosen because blue is the main colour of the default skin of MediaWiki, which Wikipedia uses. Blue is also the colour that is recognised on websites as being clickable.

When a user moves the mouse over a sentence, the sentence is highlighted with a slightly darker colour. This reinforces the notion that it is clickable, as most clickable elements on the computer screen use some kind of affirmation when moving the mouse over it. On websites, an underline is added or removed, or the link changes colour. On popup menus or even the Windows Start Menu, the elements are highlighted by changing the background colour to, again, the colour blue.

Note that links in the original article are visually preserved. What used to be a link, is now only a coloured piece of text. This is to ensure the page looks similar to the original page, and — more importantly — makes sure the user can figure out how to format links in wikitext. When keeping the cursor on a sentence for a while, the wikitext is displayed in a tooltip. This way, it is easier to understand what happens “under the hood”.

When actually clicking a sentence, a basic text box appears, along with “Save” and “Cancel” buttons. The text box contains the original wikitext. By restricting the amount of wikitext to one sentence, a user may find some “strange” codes, such as links. But because the text box is placed near the same spot the original sentence was, it is immediately recognisable and does not require any explanation. The amount of codes in the wikitext a user gets to see when editing a page using this editor, is dramatically less than when the user would use the original editor.

While this prototype clearly shows the basic concept of sentence-level editing, it still lacks basic features that are required, even for novice users. Features such as adding an edit summary, previewing changes, and some kind of copyright notice and instructions, are basic features the site (probably) cannot do without.

---

<sup>1</sup>This was our original mail to the MediaWiki community:  
<http://lists.wikimedia.org/pipermail/wikitech-1/2010-August/048821.html>.  
More information can be found here: <http://www.mediawiki.org/wiki/Extension:InlineEditor/Prototypes>.



- Navigation
- Main page
- Contents
- Featured content
- Current events
- Random article
- Interaction
- About Wikipedia
- Community portal
- Recent changes
- Contact Wikipedia
- Donate to Wikipedia
- Help
- Toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Permanent link
- Cite this page
- Print/export
- Create a book
- Download as PDF
- Printable version
- Languages
- Català
- Español
- Euskara
- Simple English

New features Log in / create account

Article [Discussion](#) [Read](#) [Edit](#) [View history](#)

## Tropical Depression Ten (2005) ★

From Wikipedia, the free encyclopedia

**Tropical Depression Ten** was the tenth [tropical cyclone](#) of the record-breaking [2005 Atlantic hurricane season](#). It formed on August 13 from a [tropical wave](#) that emerged from the west coast of [Africa](#) on August 8. As a result of strong [wind shear](#), the depression remained weak and did not strengthen beyond tropical depression status. The cyclone degenerated on August 14, although its remnants partially contributed to the formation of Tropical Depression Twelve, which eventually intensified into [Hurricane Katrina](#).<sup>[1]</sup> The cyclone had no effect on land, and did not directly result in any fatalities or damages.

Contents
<a href="#">1 Meteorological history</a>
<a href="#">2 Impact</a>
<a href="#">3 See also</a>
<a href="#">4 References</a>
<a href="#">5 External links</a>

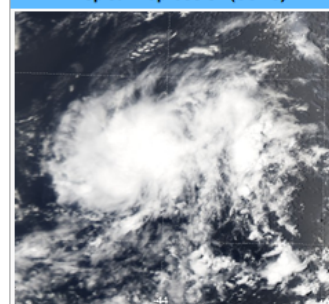
### Meteorological history [edit]



On August 8, a [tropical wave](#) emerged from the west coast of [Africa](#) and entered the [Atlantic Ocean](#). Tracking towards the west, the depression began to exhibit signs of [convective organization](#) on August 11. The system continued to develop, and it is estimated that Tropical Depression Ten

formed at 1200 UTC on August 13. At the time, it was located about 1,600 miles (2,600 km) east of [Barbados](#).<sup>[2]</sup> Upon its designation, the depression consisted of a large area of thunderstorm activity, with curved [banding features](#) and expanding [outflow](#). However, the environmental conditions were predicted to quickly become unfavorable.<sup>[3]</sup> The depression moved erratically and slowly towards the west, and [wind shear](#) inhibited any significant intensification. Late on August 13, it was "beginning to look like [Irene](#)-junior as it undergoes southwesterly mid-level shear beneath the otherwise favorable upper-level outflow pattern".<sup>[4]</sup> The wind shear was expected to relent within 48 hours, prompting some forecast models to suggestion the depression would eventually attain hurricane status.<sup>[4]</sup>

### Tropical Depression Ten



Tropical Depression 10

<b>Formed</b>	August 13, 2005
<b>Dissipated</b>	August 14, 2005
<b>Highest winds</b>	35 mph (55 km/h) (1-minute sustained)
<b>Lowest pressure</b>	1008 mbar (hPa; 29.77 inHg)
<b>Fatalities</b>	None reported
<b>Damage</b>	None
<b>Areas affected</b>	None
Part of the	
<b>2005 Atlantic hurricane season</b>	

Figure C.1: Prototype 1.

**Awesome, you're editing Wikipedia!**

There are a few [guidelines](#) for editing an article:

- Write what you think is best for the article, or as we say here: **be bold when updating pages!** If you feel that a rule prevents you from improving Wikipedia, **ignore it**.
- **Wikipedia is an encyclopedia.** Someone else should be able to **verify** what you've written, for example in books or online.
- Write from a **neutral point of view**, and use your **own words**.

---

Can you briefly describe the changes you're making?

For example: "Fixed spelling mistake", "Corrected facts", "Wrote a new paragraph", etc.

---

**When you're done, don't forget to publish the page!**

When you click "Publish", you agree to the [Terms of Use](#). This means that you agree to share your contributions under a free license.

**Publish**

## Tropical Depression Ten (2005) ★

From Wikipedia, the free encyclopedia

**Tropical Depression Ten** was the tenth [tropical cyclone](#) of the record-breaking [2005 Atlantic hurricane season](#). It formed on August 13 from a [tropical wave](#) that emerged from the west coast of [Africa](#) on August 8. As a result of strong [wind shear](#), the depression remained weak and did not strengthen beyond tropical depression status. The cyclone degenerated on August 14, although its remnants partially contributed to the formation of Tropical Depression Twelve, which eventually intensified into [Hurricane Katrina](#).<sup>[1]</sup> The cyclone had no effect on land, and did not directly result in any fatalities or damages.



Figure C.2: Prototype 2.

## C.2 Prototype 2

The second prototype adds the basic features the first prototype lacks (Figure C.2). We would even dare to claim it is good enough for use on Wikipedia. The critical change is the box at the beginning of the page. This box gives some instructions, asks for an edit summary, includes a “Publish” button, and states the “Terms of Use”.

The “edit box” should be as small as possible. It should present the most essential information, and nothing more. It should ask for nothing more but the bare minimum. We chose to include a few basic guidelines, starting with some positive reinforcement: “Awesome, you’re editing Wikipedia!”. This invites novice users to actually edit the article. After all, what they are doing is “awesome”!

The first guideline is that you are free to do as you like, as long as you are improving the site: “Write what you think is best for the article, or as we say here: be bold when updating pages! If you feel that a rule prevents you from improving Wikipedia, ignore it.”. This actually conveys two of the most important guidelines on Wikipedia: “Be bold” and “Ignore all rules”. In fact, these guidelines are part of “Fifth pillar” of fundamental principles on Wikipedia.<sup>2</sup>

Because the statement “Be bold” can be confused with using a bold typeface, this is expanded to the first sentence, including “be bold when updating pages”. The second part is exactly the statement of the guideline “Ignore all rules”, except that this statement also talks about “maintaining Wikipedia”, which

<sup>2</sup>Five pillars of Wikipedia: [http://en.wikipedia.org/wiki/Wikipedia:Five\\_pillars](http://en.wikipedia.org/wiki/Wikipedia:Five_pillars).



is not something a novice user would know about.

The second guideline says what Wikipedia is: an encyclopedia with verifiable content: “Wikipedia is an encyclopedia. Someone else should be able to verify what you’ve written, for example in books or online.” This conveys the “First pillar” and part of the “Second pillar” of fundamental principles on Wikipedia.

The third guideline presents another part of the “Second pillar”, a neutral point of view, and encourages the user to not be afraid of writing, while implicitly telling them to respect copyright: “Write from a neutral point of view, and use your own words.”

Note that all guidelines are positive in nature. Instead of telling the user not to do this and that, they tell them what you could and should do.

Next in the edit box is the edit summary. The line above the edit summary is chosen very carefully: “Can you briefly describe the changes you’re making?” Asking for “changes you *have* made” looks strange when first encountering this page. Asking for “changes you *will be* making” looks strange when the changes have actually been made. Therefore, the page asks to describe “changes you *are* making”, which is a continuous process, without defined start and end points.

The line under the text box suggests some things you can type into it: “For example: ‘Fixed spelling mistake’, ‘Corrected facts’, ‘Wrote a new paragraph’, etc.” These are not just possible inputs for the textfield, but possible *actions* when editing the page. The user is encouraged to look for mistakes, and even to write a new paragraph.

Below the edit summary is the most important button on the page: the “Publish” button. On the original edit-page, there are some problems with the “Save” button. The first problem is the caption. “Save” can mean different things: “Will it be visible for everyone or just saved for myself?”, “Will it be saved into some kind of database, waiting for approval?” “Publish” is unambiguous: it will be shown to the world.

The second problem of the original “Save” button, is its size. On the original edit page, it is hard to find the button, because it is small, and surrounded by other buttons and text.

Because this is the most important button of the page, the “Publish” button has a distinctive, approving green color. And it is big. It is the only place in the user interface design, where a button of that size with that color is used, so it cannot be missed.

Next to the button is a description: “When you’re done, don’t forget to publish the page!”. This text serves two purposes. When reading from top to bottom, the user is reminded that somewhere there is a way of publishing. When users have not already found this button, they will now. The second purpose is telling that anything you do is not final until the page is published. This way the user is invited to try some things out, as it will not be published before hitting the button.

Finally, there is a short note of the “Terms of Use”, with one sentence that tries to summarise these terms: “When you click ‘Publish’, you agree to the Terms of Use. This means that you agree to share your contributions under a free license.”

The editing is also changed a bit: when you click a sentence, the text box that opens automatically scales with its content. When you add or remove text, the text box grows or shrinks. This is a nice way of editing, and kind of what one would expect.

The “Save” button has been replaced by a “Preview” button. When hitting this button, a rotating animation shows some work is being done. In this prototype there is no actual work being done, but in the final version a request may go to the server, requesting the rendered version of the edited sentence.

When either one of the buttons is pressed when editing a sentence, the text box is hidden, and the rendered sentence reappears. When the sentence reappears, it fades from orange to the original blue color. This is a visual indication to show where it went, as the interface may jump a bit due to the intruding editor.

### Awesome, you're editing Wikipedia!

You can edit the article below, by clicking on **blue elements** in the article.

Can you briefly describe the changes you're making?

For example: "Fixed spelling mistake", "Corrected facts", "Wrote a new paragraph", etc.

When you're done, don't forget to publish the page!

When you click "Publish", you agree to the [Terms of Use](#). This means that you agree to share your contributions under a free license.

**Publish**

Edit mode:  Text  References  Images  Templates  Full editor

There are a few [guidelines](#) for editing an article:

- Write what you think is best for the article, or as we say here: **be bold when updating pages!** If you feel that a rule prevents you from improving Wikipedia, **ignore it**.
- **Wikipedia is an encyclopedia**. Someone else should be able to **verify** what you've written, for example in books or online.
- Write from a **neutral point of view**, and use your **own words**.

## Tropical Depression Ten (2005) ★

From Wikipedia, the free encyclopedia

**Tropical Depression Ten** was the tenth [tropical cyclone](#) of the record-breaking [2005 Atlantic hurricane season](#). It formed on August 13 from a [tropical wave](#) that emerged from the west coast of [Africa](#) on August 8. As a result of strong [wind shear](#), the depression remained weak and did not strengthen beyond tropical depression status. The cyclone degenerated on August 14, although its remnants partially contributed to the formation of Tropical Depression Twelve, which eventually intensified into [Hurricane Katrina](#).<sup>[1]</sup> The cyclone had no effect on land, and did not directly result in any fatalities or damages.

**Tropical Depression Ten**  
**Tropical Depression (SSHS)**



Figure C.3: Prototype 3.

### C.3 Prototype 3

The third prototype adds a few things (Figure C.3). First of all, it adds a basic editing feature: highlighting edited sentences. This is done in a yellow color, the same color the marker pen became famous for. Thus these sentences are recognised as being “marked”.

The most important change, however, is the introduction of different modes of editing. The prototype shows how the editor could be enhanced, step by step. It also shows how a user can learn wikitext. By starting with “Text”, the user gets familiar with basic concepts, like bold, italics, and links.

Note that the texts in the different tabs are just placeholders. In this case we have not thought about them very deeply, and just copy-pasted text from pages on Wikipedia.

Slowly, the user can learn more concepts: references, images, templates. You can imagine other tabs here, too. For example “Sections”, “Lists”, “Tables”, etc. When a user feels familiar enough with the wikitext syntax, it can try out the full editor. Note the wordings “full editor”. Not “old editor” or “traditional editor”. The goal is to give potential Wikipedians an easier path to becoming a frequent editor, instead of being thrown in at the deep end. For them to truly become skilled editors, they have to be able to work with the original editor, and this would not work if it is seen as “old”.

Because the original guidelines have moved from the edit box to the “mode box”, there had to be a replacement sentence, to keep the edit box visually appealing. We chose to direct the user to see the article below, in case the user simply did not notice it was there, or has a low-resolution screen. Also, it briefly explains the concept of editing: “You can edit the article below, by clicking on blue elements in the article.”

The limitations of this kind of editor also become clear in this prototype. In this page there is one inline template: a template to convert units. It is used in the sentence “At the time, it was located about 1,600 miles (2,600 km) east of Barbados.” The middle part of the sentence is actually generated by the wikitext `{{convert|1600|mi|km}}`. At this moment, We are not quite sure if and how it will be possible to edit this sentence. In the next phase of my research we will look into this.