Editing 2.0

MediaWiki's upcoming visual editor and the future of templates

Brion Vibber

2011-08-05 Haifa
Part I: Editing
So pretty!

Wikipedia articles can include rich formatting, beyond simple links and images to complex templates to generate tables, pronunciation guides, and all sorts of details.
But when you push that “edit” button, you often come face to face with a giant blob of markup that’s very hard to read. Here we can’t even see the first paragraph of the article until we scroll past several pages of infobox setup.
Even fairly straightforward paragraphs start to blur together when you break out the source. The markup is often non-obvious; features that are clearly visible in the rendered view like links and images don’t stand out in the source view, and long inline citations and such can make it harder to find the actual body text you wanted to edit.
As many of you here today will be well aware, the way our markup displays in a raw text editor can also be really problematic for right-to-left scripts like Hebrew and Arabic. It’s very easy to get lost about whether you’ve opened or closed some tag, or whether your list item is starting at the beginning of a line. Without control over the individual pieces, we can’t give any hints to the text editor.
The same text rendered as structured HTML doesn’t have these problems; bullets stay on the right side of their text, and reference citations are distinct entities. If we could edit directly in this view, we’d save ourselves a lot of aggravation!
JanPaul Posma’s InlineEditor extension is an excellent example of work on making it easier to edit smaller pieces of a page, making the amount of other markup around you less intimidating. Though each piece is still edited as raw markup, you’re only exposed to the parts you’re working with, and you can switch them back to visual mode very quickly. Unfortunately he couldn’t make it here to make his own presentation, but his thesis paper is available online and will be good reading for some of us folks. :) One of the trickiest bits of implementation is figuring out which HTML maps to which source, and consistently mapping it back in after editing.
Wikia’s rich-text editor plugin, which has been running in production on many of Wikia’s newer sites for a while, modifies MediaWiki’s existing parser to add some structural metadata. This allows a rendered view to be edited quite nicely in an HTML editor extending FCKEditor, then translated back to wiki markup for saving.
But the complexity and fragility of the current parser makes this error-prone; some structures can’t be represented in the HTML output, forcing a fallback to plain-text editing for markup seen in many Wikipedia articles. Wikia has been hitting against the difficulties of maintaining this system further, and has already leant one of their engineers to the R&D stage of our new editor project to ensure that we can share a common next-generation visual editor.
Magnus Manske’s WYSIFTW system takes a different approach, using a custom parser and producing a view of the page that’s explicitly tailored for editing. Comments are visible, complex templates and references are folded by default but can be expanded, and infobox templates are presented as sort of editable data tables. While the current proof of concept doesn’t scale well to large pages, there are a lot of ideas we expect to lift directly from it such as a slightly-different editing-oriented view and explicit support for editing template usages.
So what are we going to make for you?
We’d love to combine the “best of both worlds” of WYSIWIG and locality-based editing. If the rendered page and editing system understand the document structure, we jump straight into a table cell, template, or special markup construct and just start editing it in-place. This is a demo I whipped up embedding a syntax-highlighting code editor to modify source code examples inline on a fully rendered article -- alas it doesn’t yet know how to actually save. :)

```javascript
// Simple examples

A simple recursive function:

```javascript
function factorial(n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

A simple personalised greeting script:

```javascript
var name = prompt("What is your name?");
alert("Welcome " + name);
```

Anonymous function (or lambda) syntax:

```javascript
function add(i, j) {
    var addLambda = function(x, y) {
        return x + y;
    };
    return addLambda(i, j);
}
```

Closures:

```javascript
function displayClosure() {
    var inc = makeIncreaser(1);
    inc(); // returns 1
    inc(); // returns 2
    inc(); // returns 3
}

function makeIncreaser(init) {
    var count = init;
}
```
Wikimedia’s Trevor Parscal and Wikia’s Inez Korczynski have been working on an editing surface that handles layout, typing, and cut-n-paste details in a more hands-on way than many other editors that simply build on browsers’ basic HTML edit area support. This lets the editor understand a custom document structure natively, which can be mapped back to the original wikitext source.
Another advantage is that, unlike systems building on browsers’ content-editable system, this works on mobile browsers like iOS and Android devices that have traditionally not handled wysiwig editing on the web well. With mobile as a key outreach priority, the possibility of an attractive mobile-optimized editing view for smartphones -- which will soon be a huge portion of all phones -- has obvious good points.
Another thing we’re trying to bake into the editor is infrastructure to support multiuser editing. In the Wikimedia community & staff alike we’ve found a huge usefulness to using tools like Etherpad to let multiple people work on shared notes at the same time... but we then have to copy them over to a wiki manually, or keep the separate links around. Neil Kandalgaonkar’s been experimenting with embedding Etherpad as a plugin as a temporary solution, and ensuring that the editor’s internal structures are suitable for the same kind of multiuser updates and transactional reordering that will let us extend from a single user with an undo stack to multiple users editing together.
Part II: Parser
So what is the parser? IT’S THE SCARY PART that turns all your scary markup into pretty web pages and structured data like lists of which links and templates are used.
The most immediate reason is of course all this editing stuff we’ve been talking about. An editor can become much more powerful and flexible when it understands the structure of the document. Having a consistent way to build and represent that structure from the markup helps a lot!
Remember the Wikia RTE limitations? The new parser will ensure that even when we can't represent everything 100% in the HTML output, we *can* send a structure that the editor understands.
Open content is a neologism coined by David Wiley in 1998 [1] which draws an analogy between open source practices and the publishing of content online. [2] Open content describes thus any kind of creative work, or content, published under an open content license that explicitly allows copying and modifying of its information by anyone, not exclusively by a single organization, firm or individual.

http://en.wikipedia.org/wiki/Open_content

Then, there’s a Big-Picture reason. Wikipedia is an open-content project, with an explicit goal to create documents that can be redistributed and re-used. This is dependent on being able to use the data in compatible ways. MediaWiki itself is free software, but it’s not suitable for all uses -- our data must be able to live on separately.
There are dozens of alternate implementations of MediaWiki’s syntax, but none of them behaves quite the same way MW does. Most will fail dramatically on complex articles and templates.

http://www.mediawiki.org/wiki/Alternative_parsers
Hidden behind even simple-looking text -- not to mention complex infoboxes -- can lie all sorts of templates, parser functions, and funky behavior.
mwlib, the parser used by our PediaPress–powered PDF & print on demand export system, handles most constructs fairly well, but may still have rendering issues.
Despite very good overall syntax coverage, the Sweble wikitext parser doesn’t handle a lot of template and parser function constructs, and can simply break on them.
Why so hard?
The parser’s main body consists of layer upon layer of fragile string processing, leading to strange bugs when different layers fail to respect each other properly. More structured processing should make this less error-prone.
Templates that separately open and close lists and tables are very frequently used, such as these sports medals templates on English-language Wikipedia.
Here, each template expands into just a piece of the table. If we expand the templates in-place in a node tree, the table rows will be siblings or cousins of the table-open node, rather than its children. Because HTML tables require a hierarchical structure, we need to construct a separate wiki ‘DOM’ and reassemble some larger structures into a legit HTML output tree.
I’ve started on a JavaScript–based parser implementation that can be used in a client–side Gadget for testing on existing wikis. Later on we’ll build a PHP, and possibly an accelerated C++ version to integrate into MediaWiki core; having two compatible implementations will also give us more confidence in the specification that we’re creating. Trevor and Inez are working on the visual editor components, which will start to plug into the actual parser output soon. Neil’s also putting together the infrastructure we’ll need to do multi–user editing — we’ve seen from Wikimedians’ use of Etherpad how useful this is, and consider it a long–term requirement for a rich editor.
The ParserPlayground extension carries our in-progress parser layer, currently using a basic Parser Expression Grammar parser generator and an ad-hoc JSON-style intermediate format. This can also be imported as a gadget on live sites; as we continue integrating pieces, it’ll be available for anyone to try out on real Wikipedia and Wikibooks, to test rendering and editing in a real environment.

---


The ParserPlayground extension carries our in-progress parser layer, currently using a basic Parser Expression Grammar parser generator and an ad-hoc JSON-style intermediate format. This can also be imported as a gadget on live sites; as we continue integrating pieces, it’ll be available for anyone to try out on real Wikipedia and Wikibooks, to test rendering and editing in a real environment.
To make it work as a gadget, the initial work on the new parser is being done in JavaScript which can be loaded in safely without changing the server configuration. This will also guarantee that once we finish the PHP code for core we’ll have two provably compatible implementations of our specification. Creating a clear interface between the parser and its host application environment is an important part of this -- MediaWiki’s parser has traditionally been hard to extract out to reuse even in other PHP–based software.
Of course we’re not just going to rely on humans for testing! The JavaScript parser implementation can be batch-tested in real browsers through the TestSwarm system, and in a (much faster!) command-line server environment using Node.js, a server-side environment built around Google’s V8 engine.
I’ve started on a batch test system that runs over an entire Wikipedia data dump: our production sites can become a testing corpus to look out for regressions and help us determine when the new parser is “good enough” to cover all but the needed corner cases.
So you’re probably all asking yourselves, WHEN IS THIS COMING? When can I start helping out? We hope to have opt-in testing of a real, if basic editor by the end of the year, moving towards more public rollouts in mid-2012. Brave gadget testers should have some fun things to play with even earlier this year.
If you’re very brave though, you can hop into the earlier work we’re doing now: defining the environment and interface for the parser, the intermediate data structures, and the creation of the basic editing widget.
What about templates?

So, what about templates? We already gave some examples of tricky constructs that the parser & editor will have to deal with -- the intention is to get a solidly defined implementation that’s very nearly compatible with the previous parser so it can replace it for all our current data structures. So the good news is, you shouldn’t have to change too many templates to make them work; we’ll try our best to make the parser work with them.
But we know that these kinds of templates are just *really hard to read*, write, and maintain. So we’re also keeping in mind new ways to create templates in the future.
While this work should be largely independent of the editing work, having a clean document structure that we could pass into more programming-like templates could make some kinds of things a LOT easier to maintain. For instance anything that has to loop over multiple inputs.
It's still up in the air exactly what we'll end up with, but expect some experimentation with JavaScript and/or simple languages resembling a JavaScript subset. Cleaner interfaces, structured data, better caching, etc could end up allowing much more efficient template execution, including offloading slow processing to a dedicated script engine.
Of course that leads on to other projects to help contributors create and share their own interactive code; making Gadgets and user scripts easier to create and safer to share by defining stable JavaScript interfaces. Hopefully I’ll be able to do a talk on that at next year’s Wikimania!
the

http://www.mediawiki.org/wiki/Future

is now!

end :D