web-mode.el
Heterogeneous recursive code parsing with Emacs Lisp

François-Xavier Bois
Kernix Lab
15 rue Cels
75014 Paris - France
+33 (1) 53 98 73 43
fxbois@kernix.com

ABSTRACT

web-mode.el is an Emacs module for editing HTML templates. Unlike other “source codes”, web templates can include various language components (*heterogeneous dimension*) that may embed themselves into each other (*recursive dimension*).

Indeed, an HTML document may contain a JavaScript that embeds a PHP block.

```
<div>
  <h1>title</h1>
  <script>var x = <?=$x?>;</script>
</div>
```

This recursive aspect invalidates standard ways of lexing (syntactic tokens), highlighting (colorizing) and indenting.

All the power of Emacs is necessary to enable contextual indentation and highlighting.

This paper describes web-mode.el features and some of its internal processes.

Categories and Subject Descriptors

Coding Tools and Techniques, Document and Text Editing, User Interfaces, Programming Environments, Reusable Software

General Terms

Algorithms, Languages.

Keywords

Emacs, Lisp, Web, HTML, templates, engines.

1. OVERVIEW

Emacs owes a big part of its success to two strengths that characterize it: its extensibility through modules (modes) and its “Lisp Machine” dimension.

Though compatible with the vast majority of programming languages, Emacs has suffered in the past years of its relative weakness in the field of web template editing. Those HTML documents which embed parts written in different languages (JavaScript, CSS, PHP, Java, Ruby, etc) are at the core of web development, a very dynamic domain of computer science.

1.1 Multi modes approach

HTML template editing under Emacs has long been the prerogative of “multi modes” like mumamo.el, mmm-mode.el or multi-web-mode.el. Those modes rely on the “available” dedicated modules to handle their corresponding “parts” in the template.

A template with HTML, JavaScript, CSS and PHP content may for example use

- nxml.el for HTML
- js2-mode.el for JavaScript code (located between `<script>` and `</script>` tags)
- css-mode.el for styles (between `<style>` and `</style>`)
- php-mode.el for PHP statements delimited by `<?php` and `?>`

In order to explain how this “multi modes” approach works, one should keep in mind that Emacs modes are divided in two families: the major and minor modes.

A major mode handles the responsibility of buffer highlighting and indentation; only one major mode may be running for a buffer at a given time.

To let many major modes “coexist”, a “multi mode” (which is a minor mode) loads a specific major mode according to the cursor position `(point)`. If the cursor is between the delimiters `<?php` and `?>`, the “multi mode” activates the major mode php-mode.el. The *narrowing* mechanism helps the “multi-mode” to restrict the working space to the region between the delimiters (`<?php` and `?>`).

This “Unix-like” approach is very appealing:

- Each mode has one role that it tries to achieve the best it can.
- By combining those modes one may achieve a larger ambition.

Alas it can be very frustrating for the users:

- Going back and forth between modes triggers visual artifacts and slows down the workflow.
- Incompatibilities between modes result conflicts and errors.
- Customization is difficult and inconsistent (indeed, each major mode has its own parameters).
- The lack of common ground prevents advanced features (no context is shared between the major modes).

1.2 web-mode.el

Aware that no satisfactory results would ever happen with this approach, I started in 2011 the development of the major mode web-mode.el. The main features of this mode are

- Autonomous (no other major mode is required)
- Fast
- Simple (no configuration required)
- Effective (auto-closing, code folding, tag navigation)
- Aware of all the HTML format specificities
- Compatible with more than twenty engines (erb, jsp, php, asp, razor, django, mason, etc.)

2. IMPLEMENTATION NOTES

2.1 Terminology

As was said previously, a template may embed various code components. Two kinds will be considered here:

- A *part* is interpreted by the navigator (e.g. a JavaScript part or a CSS part).
- A *block* is processed (client-side or server-side) before being rendered by the navigator e.g. a PHP block, an Erb block, a dustjs block etc.

This terminology is not a standard but is useful for function/variable naming and for the documentation.

2.2 The main loop

Two tasks are executed as soon as the buffer is loaded and altered:

- Lexing: to detect the various entities (blocks, parts, nodes) and for each one, identify their tokens: block/part strings/comments, node attributes, block delimiters.
- Highlighting: to colorize the code.

2.3 Lexing phase

Three steps are necessary for parsing the code

1. Block identification and scanning.
2. HTML scanning (HTML tags / attributes / comments, doctype declaration)
3. Parts scanning (JavaScript, CSS).

When a file is opened, web-mode.el scans the entire buffer. Subsequent scans are performed on smaller zones “around” the altered zone: it is called the invalidation phase and is described below.

The order of the steps is very important: indeed, nodes/parts: identification must be done outside the blocks.

```
<div>
  <?php /* <script>var x = 1</script> */ ?>
</div>
```

Part scan is done during a specific step because web-mode.el can be used to edit files whose content-type is not HTML but JavaScript or CSS. For example *.js.erb is a Ruby on Rails JavaScript template.

Given the large variety of tokens (specific to languages) and the recursive dimension of code, web-mode.el can’t rely on Emacs internal functions to tokenize and highlight.

Parameters are associated to each engine family and are used for tokenizing, indenting, highlighting, auto-pairing and auto-closing.

With each engine are associated

- Delimiters (e.g. `<?php ?>`)
- Control blocks (e.g. `<?php if(): ?>`)
- Syntactic tokens regular expressions
- Auto-pairs, snippets

In order to parse the code, web-mode.el must know which is the engine associated with the template. This association is automatic as soon as the file extension is obvious (e.g. *.erb). Association must be forced when the extension is too common.

```
(require 'web-mode)
(add-to-list 'auto-mode-alist
             '("\\.html\\'" . web-mode))

(setq web-mode-engines-alist
      '(("erb" . "/rails/.*\\.html\\'")
        ("php" . "/zend/.*\\.html\\'"))
)
```

2.4 Custom text properties

The scanning/lexing phase will help store information that will be used to implement interactive features like indentation, folding, or tag navigation.

This process is achieved with the “text-properties” which are plists attached to every single character in the buffer.

web-mode.el adds to the common properties (e.g. `'face`, `'visibility`) new ones that describe the following states

- `'block-side` is set to `t` throughout blocks characters
- `'block-(beg|end)` mark the blocks boundaries
- `'block-token`[1] is `'string`, `'comment` or `'delimiter` on block tokens
- `'block-controls` is the current block a control block? (e.g. `{% for %}` … `{% endfor %}`)
- `'tag-type` tells if the tag is a `'start`, `'end` or `'void` tag (`'tag-name` store the tag name).

Bitmask on `'*-beg` properties is an additional way to store information useful for the highlighting phase.

2.5 Indentation

As the main purpose of web-mode.el is to remain autonomous, a generic indentation engine was developed.

web-mode.el deals with three kinds of indentations

1/ HTML indentation relies on finding the first previous line beginning with a start tag. Deciding if the current line should be indented is done by counting start / end tags and see if a start tag remains unclosed.

[1] All the 'block-* properties are available as 'part-* equivalents.

2/ Bracket based indentation. It relies on counting the number of unclosed brackets. Inline calls are also handled by looking at the first unclosed bracket. This kind of indentation is used by languages like PHP, JavaScript, CSS, etc.

3/ With stack based indentation, each indentation line depends directly on the current and previous lines. For instance, if the previous line is a control statement (`if`) the current line is indented.

It is important to remember that templates are at the core of the MVC design pattern. The developer must separate the various logical components of its application: Model, View (templates) and Controller. The more code is placed in the Model or Controller components, the better.

More generally, a good habit in web development is to put foreign code in specific files (e.g. *.css for styles, *.js for javascripts, *.php for engine statements). Thus the indentation engine should not have to deal with complex and large parts or blocks.

2.6 Consistency

Being able to consider the whole buffer state is very useful to implement advanced features. The markup indentation engine can for example evaluate HTML elements and control blocks when calculating the indentation offset.

```
<div>
  <?php if ($x): ?>
    <span></span>
  <?php endif; ?>
<div>
```

2.7 Highlighting

Highlighting a buffer in Emacs involves the font-locking mechanism. The recursive dimension of templates and some complex parsing rules (e.g. for HTML attributes) prevents the use of standards font-lock keywords.

As for the scanning phase, the highlighting phase involves three steps:

1. node highlighting
2. part highlighting
3. block highlighting

Ending with block highlighting reflects a logical situation: a block can be included in a part or a node, block highlighting is thus priority.

Two techniques are used for highlighting

- Direct setting of the `'font-lock-face` text-property for HTML nodes (brackets, tags, attributes)
- Font-locking keywords for parts and blocks.

2.8 Decoration

After the highlighting phase, web-mode.el may “decorate” some of the tokens:

- String: variable interpolation (for erb and php, in double quoted string), css colorization (background refects the css color).
- Comment: keyword highlighting (ex. TODO, FIX).

2.9 Invalidation

Most major modes delegate “region invalidation” to Emacs. This process is the responsibility of font-locking; it automatically detects syntactic tokens and refreshes the colors of the altered zone.

As web-mode.el scans the buffer by itself, it has to trigger a new scan as soon as the user alters the buffer. The `'after-change-function` hook is used for this purpose. To ensure that parts and blocks are properly scanned, the following rule has been set: the region should begin and end with an HTML tag.

For the highlighting phase, the same region should be considered. web-mode.el can influence font-lock by associating a custom function to the `'font-lock-extend-region-functions`.

One should note that the lexical invalidation must be done before the highlighting; indeed highlighting uses some of the text-properties set by the lexical process (`'tag-attr`, `'block-token`, `'part-token`, etc.)

3. CONCLUSION

Thanks to the power of Lisp and to the advanced Emacs mechanisms, web-mode.el is able to provide a very robust and rich experience to its users.

Invalidation of a zone located in a part or a block is still a flaw that needs to be addressed. Indeed when such a part or block is huge, re scanning and re highlighting it entirely can be pricy. Identifying a narrower zone inside the block (or the part) is a very difficult task whenever this process must work with many languages/engines.

4. ACKNOWLEDGMENTS

A special thanks to Stefan Monnier a great Emacs maintainer and a wonderful guide to the Emacs internals.

5. REFERENCES

[1] François-Xavier Bois. *web-mode.el presentation and documentation*. http://web-mode.org.

[2] François-Xavier Bois. *web-mode.el code repository*. https://github.com/fxbois/web-mode.

[3] James Clark. *nXML mode, powerful mode for editing XML documents*. http://www.thaiopensource.com/nxml-mode.

[4] Multi Modes. *Introduction to multi modes*. http://www.emacswiki.org/emacs/MultipleModes.