ocamlweb: a literate programming tool for Objective Caml

Jean-Christophe Filliâtre and Claude Marché
http://www.lri.fr/~filliatr/ocamlweb

Contents

1 Introduction

Literate programming has been introduced by D. E. Knuth in 1984. The main idea is to put the code and its documentation in the same file and to produce from it a document which is readable by a human, and not only by a machine. Although ocamlweb borrows a lot of ideas from Knuth's original tool (called WEB), there are big differences between them. First, WEB allows you to present the pieces of your code in any order, and this is quite useful when using poorly structured languages, like Pascal or C. But Objective Caml is already highly structured, and this is no more useful. Moreover, WEB requires the use of a tool to produce the code from the WEB file, which greatly complicates the use of your favorite source-based tools (dependencies generator, debugger, emacs mode, etc.). When using ocamlweb, the documentation is inserted in the code as comments (in the Caml sense), and your code is not linked to the existence of ocamlweb in any way.

Currently, the task of ocamlweb may be seen as:

1. making a nice document with the code and its documentation;

2. generating a global index of cross-references, where each identifier is associated to the lists of sections where it is defined or used.

2 Principles

Documentation is inserted into Caml files as *comments*. Thus your files may compile as usual, whether you use ocamlweb or not. ocamlweb presupposes that the given Caml files are well-formed (at least lexically, but also syntactically if you want the cross references to be correct). ocamlweb understands each Caml file as a sequence of paragraphs, each paragraph being either a piece of code or a piece of documentation. Documentation starts and ends with Caml comment's delimiters `(*` and `*)`, and is a regular LaTeX text. Code starts with any characters sequence other than `(*` and ends with an empty line.

Two styles are available for the final document:

- a WEB style, where the code and documentation are displayed in *sections*, numbered consecutively starting from 1. A new section is started at the beginning of each file. You can also start a new section manually (see the paragraph below about controls);

- a LaTeX style, where you structure your document as you like, using the usual sectioning commands of LaTeX.

Escapings: code inside documentation and vice versa. The first feature you will require is the ability to quote code inside documentation and conversely. The latter, namely documentation inside code, is simply obtained by normal Caml comments filled in with a LaTeX contents. The former is obtained by quoting code between the delimiters `[` and `]`. Square brackets may be nested, the inner ones being understood as being part of the quoted code (thus you can quote a list expression like $[1; 2; 3]$ by writing `[[1;2;3]]`). Inside quotations, the code is pretty-printed in the same way as it is in code parts.

Controls. In addition to the default behavior of ocamlweb, you can control it through a small set of commands, which are comments of a particular shape. These commands are the following:

`(*s`

Starts a new section. (Meaningless in LaTeX style.)

`(*i  ...  i*)`

Ignores all the text between those two delimiters. Such "comments" cannot be nested but any Caml code and comments may appear between them, included nested Caml comments. You can use those delimiters to enclose a comment that shouldn't appear in the document, but also to hide some Caml code, putting it between `(*i*)` and `(*i*)` in such a way that your code still compiles but is ignored by ocamlweb. (Notice that if you use the delimiters `(*i` and `i*)` directly, the text enclosed is commented for both Caml and ocamlweb.)

`(*c`

Tells ocamlweb that this is a real Caml comment an not a documentation text. Mainly useful is you want to insert comments after empty lines. However, this is not the spirit of ocamlweb, and you are encouraged to put the documentation in separate LaTeX paragraphs and not in traditional Caml comments.

`(*r`

Justifies an end-of-line comment on the right margin.

`(*p ... *)`

Insert some material in the LaTeX preamble. See also command line option `-p`.

Pretty-printing. ocamlweb uses different faces for identifiers and keywords, and use mathematical symbols for some Caml tokens. Here are the correspondences.

`->`	$\rightarrow$	`<-`	$\leftarrow$	`*`	$\times$
`<=`	$\leq$	`>=`	$\geq$	`~-`	$-$
`<>`	$\neq$	`==`	$\equiv$	`!=`	$\not\equiv$
`or`, `||`	$\vee$	`&`, `&&`	$\wedge$	`not`	$\neg$

Integers and floating-point literals are pretty-printed like this:

`123`	$\rightarrow$	123
`0b010011`	$\rightarrow$	010011_2
`0o466`	$\rightarrow$	466_8
`0x3fff`	$\rightarrow$	$\mathtt{3fff}_{16}$
`1.2e6`	$\rightarrow$	$1.2 \cdot 10^6$
`1e-4`	$\rightarrow$	10^{-4}

Characters strings are pretty-printed in fixed-width font, and spaces are explicitly shown as the symbol ␣, as in `"a␣constant␣string"`.

3 Hints to get a pretty document

ocamlweb is rather robust, in the sense that it will always succeed in producing a document, as soon as the Caml files are lexically correct. However, you may find the result rather ugly if so were your source files. Here are some style hints to get a pretty document.

First, ocamlweb is not a real code pretty-printer: it does not indent your code, and it does not even cut your code lines when they are too long. The code will always appear formatted as it is in the source files. The indentation at the beggining of each line is output as a proportional space (tabulations are correctly translated). In particular, if you want your pattern-matchings to be aligned, you have to put a '`|`' also in front of the first pattern. Here is the difference:

$\mathsf{let}\ f\ =\ \mathsf{function}$	$\mathsf{let}\ f\ =\ \mathsf{function}$
$\quad O\ \rightarrow\ \ldots$	$\quad \mid O\ \rightarrow\ \ldots$
$\quad \mid (S\ p)\ \rightarrow\ \ldots$	$\quad \mid (S\ p)\ \rightarrow\ \ldots$

4 Usage

ocamlweb is invoked on a shell command line as follows:

`ocamlweb` < *options and files* >

Any command line argument which is not an option is considered to be a file (even if it starts with a `-`). Caml files are identified by the suffixes `.ml` and `.mli`, and LaTeX files by the suffix `.tex`. The latter will be copied 'as is' in the final document. The order of files on the command line is kept in the final document.

Command line options

`-o` ***file***, `--output` ***file***

Redirects the output into the file '*file*'.

`--noweb`

In that case, there are no sections à la WEB, and the user structurates his document directly through LaTeX commands, like for instance `\section`, `\subsection`, etc. There is still an index, in the usual LaTeX sense. (Therefore, if you have introduced LaTeX sections and subsections, their numbers will appear in the index.)

`-s , --short`

Do not insert titles for the files. The default behavior is to insert a title like "Module Foo" or "Interface for module Foo" for each file.

`--no-index`

Do not output the index.

`--dvi`

Output in DVI format, instead of LaTeX format. (You need `latex` to be installed and present in your path.)

`--ps`

Output in PostScript format, instead of LaTeX format. (You need both `latex` and `dvips` to be installed and present in your path.)

`--html`

Output in HTML format, instead of LaTeX format. (You need both `latex` and `hevea` to be installed and present in your path.)

`--hevea-option` ***option***

Passes the given option to `hevea` (to be used with the `--html` option). It is mainly useful to specify where the file `ocamlweb.sty` is to be found, when not installed in the `hevea` library directory, using `--hevea-option "-I` *dir*`"`.

`--extern-defs`

Keeps the external definitions in the index i.e. the identifiers which are not defined in any part of the code. (The default behavior is to suppress them from the index, even if they are used somewhere in the code.)

`--header`

Does not skip the header of Caml files. The default behavior is to skip them, since there are usually made of copyright and license informations, which you do not want to see in the final document. Headers are identified as comments right at the beginning of the Caml file, and are stopped by any character other then a space outside a comment or by an empty line.

`--no-preamble`

Suppresses the header and trailer of the final document. Thus, you can insert the resulting document into a larger one.

`-p` ***string***, `--preamble` ***string***

Insert some material in the LaTeX preamble, right before `\begin{document}`. See also the control `(*p`.

`--latex-option` ***option***

Passes the given option the LaTeX package `ocamlweb.sty`.

`--impl` ***file***, `--intf` ***file***, `--tex` ***file***

Considers the file '*file*' respectively as a `.ml` file, a `.mli` file or a `.tex` file.

`--files` ***file***

Read file names to process in file '*file*' as if they were given on the command line. Useful for program sources splitted in several directories. See FAQ.

`--no-greek`

Disable use of greek letters for single-letter type variables. For example, the declaration of `List.hd` is displayed as

$$\mathit{List.hd} : \alpha \ \mathit{list} \to \alpha$$

but with this option, it will be displayed as

$$\mathit{List.hd} : \ '\!a \ \mathit{list} \to \ '\!a$$

`-q`, `--quiet`

Be quiet. Do not print anything on standard error output except errors.

`-h`, `--help`

Gives a short summary of the options and exits.

`-v`, `--version`

Prints the version and exits.

5 The `ocamlweb` LaTeX style file

In case you choose to produce a document without the default LaTeX preamble (by using option `--no-preamble`), then you must insert into your own preamble the command

`\usepackage[`*options*`]{ocamlweb}`

Alternatively, you may also pass these options with the `--latex-options` option of the `ocamlweb` command.

The options that you may pass to the package are the following:

`noweb`

Tells `ocamlweb.sty` that the document was generated with the `--noweb` option so that no WEB sections are used, and the index was generated by referencing the LaTeX sections, subsections, etc.

`novisiblespaces`

By default, spaces in strings of CAML code parts are output as ␣. They will be output as real spaces if you select this option.

`bypages` / `bysections`

When not in WEB sectioning style, these options specify whether the index must refer to page numbers or section numbers. `bypages` is the default. In WEB sectioning style, this option has no effect.

Additionally, you may alter the rendering of the document by redefining some macros:

`ocwkw`, `ocwbt`, `ocwupperid`, `ocwlowerid`, `ocwtv`

The one-argument macros to typeset keywords, base types (such as `int`, `string`, etc.), uppercase identifiers (type constructors, exception names, module names), lowercase identifiers and type variables respectively. Defaults are sans-serif for keywords and italic for all others. Some of the single-letter type variables are displayed by default as greek letters, but this behaviour can be selected by the `--no-greek` option.

For example, if you would like a slanted font for base types, you may insert

```
\renewcommand{\ocwbt}[1]{\textsl{#1}}
```

anywhere between `\usepackage{ocamlweb}` and `\begin{document}`.

`ocwlexkw`, `ocwlexident`, `ocwyacckw`, `ocwyaccident`

Analogous macros as above, to typeset lex keywords, lex identifiers, yacc keywords and yacc identifiers respectively.

`ocwinterface`, `ocwmodule`, `ocwinterfacepart`, `ocwcodepart`

One-argument macros for typesetting the title of a `.mli` file, the title of a `.ml` file, the interface part of a `.ml` file and the code part of a `.ml` file, respectively. Defaults are is

```
\newcommand{\ocwinterface}[1]{\section*{Interface for module #1}}
\newcommand{\ocwmodule}[1]{\section*{Module #1}}
\newcommand{\ocwinterfacepart}{\subsection*{Interface}}
\newcommand{\ocwcodepart}{\subsection*{Code}}
```

and you may redefine them using `\renewcommand`.

6 FAQ

1. **What about an HTML output?**

   Use hevea, the LaTeX to HTML translator written by Luc Maranget (freely available at `http://pauillac.inria.fr/hevea/`), with the following command-line:

   hevea `ocamlweb.sty` *file.tex*

   where *file.tex* is the document produced by ocamlweb. The package `ocamlweb.sty` contains the necessary support for hevea.

2. **How can I customize the appearance of the final document?**

   You can redefine some of the LaTeX macros of the `ocamlweb.sty` style file. See Section 5. We do not recommend to modify the `ocamlweb.sty` file directly, since you would not be able to update easily to future versions. You should rather redefine the macros using `\renewcommand`. If you want to customize other parameters that are not currently customizable, please contact the developers.

3. **How can I insert 'usepackage' commands, or whatever else, in the LaTeX preamble?**

   Use the option `-p` or the corresponding control `(*p`.

   If you really want a different LaTeX preamble, for instance to use a LaTeX class other than `article`, then use the option `--no-preamble` and catenate the result with your own header and trailer (or use a `\input` or `\include` command to insert the file generated by ocamlweb in your main LaTeX file.)

4. **How can I use square brackets in LaTeX, since they are reserved characters for ocamlweb?**

   There is no escape sequence for the moment. But there is an easy solution to this problem: define macros for your LaTeX notations involving square brackets and put them in the preamble or in a `.tex` file that you will provide on the ocamlweb command line. Then, there is no longer square brackets in your source comments, but only calls to these macros.

5. **What about lexers and parsers?**

   There is support for ocamllex lexers and ocamlyacc parsers since version 0.9. They are recognized by their suffix on the command line (do not use option `--impl` anymore).

6. **I would like my pattern-matching right-hand sides to be aligned. Is it possible?**

   No, it is not, since ocamlweb uses proportional fonts. But you can still align your pattern-matching in your source files, since ocamlweb converts multiple spaces into single ones.

7. **What can I do with a large program with sources in several directories, organised as libraries?**

   First alternative: you can run an ocamlweb command in each directories, producing a documentation for each library. If you want to have the documentations alltogether in the same TeX file, run each ocamlweb commands with option `--no-preamble` to avoid generation of the LaTeX preamble, build by hand a master document file with the preamble you want, without forgetting the `\usepackage{ocamlweb}`, and input each file in each directory with the `\input` or the `\include` macro.

   Second alternative: the main problem with the first alternative is that you will have an separate index for each libraries. If you want a global index, you need the run only one `ocamlweb` command. If you don't want to put all source file names on one single command line, you can use the `--files` option which allows you to read the source file names from a file. Example :

   ```
   ocamlweb --files lib1/source-files --files lib2/source-files
   ```

   will produce documentation for files in `lib1` and `lib2` provided that the files `source-files` in each directory contain the name of the documented source files.