Literate `plain` Source is Available!

Włodek Bzyl
Instytut Matematyki
Uniwersytet Gdański
Wita Stwosza 57
80–952 Gdańsk
Poland
Email: `matwb@univ.gda.pl`

Abstract

Based on Norman Ramsey's NOWEB system a new literate tool for the TEX language has been built. The new system was used to create a 'literate `plain`' source. Although the resulting file is principally `plain.tex` code interleaved with documentation, borrowed mainly from *The TEXbook*, it presents the whole code from a different perspective. The documentation is organized around the macros as they appear in the `plain.tex` file rather than around the topics as in *The TEXbook*. This means that the typeset `plain.dvi` is *not* a user manual, even though many notions are explained there.

Introduction

When it was introduced, literate programming was synonymous with `WEB`, a system for writing literate Pascal programs. Since then many different `WEB`s, each aiming at a particular programming language (or small group of related languages), have been created. Each `WEB` is constructed of two separate parts, one called `TANGLE`, the other `WEAVE`. Typically each part consists of just one program performing many tasks — it expands macros, prettyprints code, generates and sorts an index, etc. This makes adaptation of the existing `WEB` to another language extremely difficult.

Another approach to literate programming was presented by Norman Ramsey, in NOWEB. He designed and realized the `TANGLE/WEAVE` pair as UNIX pipes. By extending and/or replacing parts of pipes with programs, written in AWK, ICON, Flex, Perl, C, TEXor METAFONT, a new tool could be created with relatively small effort. As a result, with NOWEB, it was possible to create a simple `TEX-WEB` system by writing an AWK script and a new TEX format.

`WEB` for everyone?

`WEB` is a powerful tool. The strength of literate programs lies in their ability to produce high-quality typeset documentation. The strength of literate programming lies in allowing you to write code where you are telling humans what the computer should do, instead of telling computer what should be done. Obviously we are more efficient and precise when communicating with humans than computers. Thus literate programs are more easily written and maintained than ordinary ones.

`WEB` is a complex tool. A literate program consists of pieces of documentation and named chunks containing code and references to other chunks. The pieces are arranged in an order which helps to explain (and understand) the program as a whole. The `WEB` system consists of two processors: `TANGLE` and `WEAVE`.

`TANGLE` is used to extract a program by replacing one named chunk by its definition. The process of replacement is recursive; it continues until no named chunks remain. From one `WEB` source many programs could be extracted (by presenting `TANGLE` with different chunks).

`WEAVE` is used to convert `WEB` markup into TEX markup as described and coded in a separate format file. It handles numerous typographical details of typeset documentation and provides support for typical tasks such as cross-referencing, preparation of indexes, bibliography. Formats for long and short documents will be different. To typeset a converted file you will need TEX running on your system. Errors can creep into TEX code. Getting TEX code working with other formats could end with a short trip into the TEX language (this will be needed if you plan your literate program to form part of an article, a report, or a book).

We learn by reading: why not read 'literate books'? There are a few such books already and more will appear. We learn by writing too: why not try one of the existing tools? The C/C++/Fortran programmer could try CWEB or FWEB. Programmers writing in other languages could check the CTAN directory `/tex-archive/web` for other possible tools. If your language is not on the list, or you are not able to express yourself within the style offered, then you are welcome to join the province of those who build their own tools. This territory is growing fast due to the efforts of Norman Ramsey, who established a base for creating simple and extensible literate tools.

Presenting a new tool: TEX-WEB

Norman Ramsey was the first to attempt to create a generic literate tool, not aimed at a particular language. Such a tool would (of itself) be useless because of its generality — the key to the usefulness of NOWEB lies in its extensibility. The tasks for `TANGLE` and `WEAVE` were divided among standalone programs. To simplify tangling and weaving a front end was introduced. It performs a kind of lexical analysis of the source, a task previously performed by both processors separately. The front end provided with NOWEB is called `markup` because it marks each line of source as line of text, as beginning/end of code/documentation, as definition/use of named chunks, etc.[1]

`WEAVE`

```
markup foo.tw |
       awk -f web2tex.awk > foo.tex
```

With markup as its front end, `WEAVE` was built as a pipeline where AWK, obeying commands from the script `web2tex.awk`, reads a marked source line by line and performs actions depending on the line type. Most of the time it inserts a bunch of TEX macros, for example inserting index macros.

The format `tweb.sty` provides support for cross references, indexes, and multicolumn output. There you find macros `\chapter`, `\[sub[sub]]section`, `\paragraph`[2], `\printcontents`, `\title`.

`TANGLE`

```
markup foo.tw | nt > foo.sty
markup foo.tw | nt -R'Chunk B' > foo.sty
markup foo.tw | mnt 'Chunk B' 'Chunk A'
```

Here we have several possibilities. We can extract code beginning from the chunk named '`<<*>>`', or from '`Chunk B`' (see template file below). Finally, '`Chunk A`' and '`Chunk B`' could be simultaneously extracted to the files with the same names.

TEX

```
tex foo.tex
makeindex -s dnd.ist -o foo.dnd foo.ddx
makeindex -s und.ist -o foo.und foo.udx
makeindex -s chn.ist -o foo.chn foo.chk
tex foo.tex
```

Indexes are sorted by `makeindex`. Three very short index style files provide formatting of the different indexes. (MSDOS `makeindx` breaks on large indexes.)

Sample `Makefile`. To ease work with tools a simple `Makefile` is provided. Type `make` on the command line, press the `Enter` key, and the following lines will appear on a terminal:

```
      Tangling:  make foo.sty
        Texing:  make foo.dvi
       Weaving:  make foo.tex
Making archive:  make archive
      Cleaning:  make clean or veryclean
```

Since there are many different conventions for where to store files in a file system, three variables are defined in the `Makefile`:

- `SCRIPTDIR` — where `web2tex` and other scripts are stored (defaults to `BIN`),
- `INDEXDIR` — where index styles are stored (defaults to `IDXSTY`),
- `NOWEBDIR` — where the programs `markup`, `nt`, `mnt` are stored (defaults to `/usr/local/lib/noweb`).

Also:

- `MAKEINDEX` — the name of the makeindex program (defaults to `makeindex`),

deals with the fact that the command has a different name on MSDOS systems.

Template of TEX-WEB source. The structure of a TEX-WEB file is shown in the example below.

File name: `foo.tw`

```
\title{foo.tw -- template file}
\printcontents % if you want TOC
@
The skeleton of the file foo.tw
<<*>>=
<<Chunk A>>
<<Chunk B>>
@
Documentation for Chunk A.
<<Chunk A>>=
(TEX code / references to other chunks)
```

[1] There is `unmarkup` which works in the opposite way. I also borrowed two more programs: `nt` (tangle) and `mnt` (multiple tangle) from NOWEB.

[2] These macros should not be overused. Usually the chunk name alone is a better choice.

```
@
Documentation for Chunk B.
<<Chunk B>>=
(TeX code / references to other chunks)
```

Documentation chunks begin with the line that starts with `@` followed by space or newline. Code chunks begin with <<Chunk name>>= on a line by itself. Chunks are terminated by the beginning of another chunk or end of file.

Making changes/updates. The change file mechanism is not needed in the case of the TeX language. Change files are used to incorporate system dependent code into a source file, but TeX code is already system independent: TeX code could only be 'format dependent'. Another feature of the format file is that it evolves with time, but the intermediate versions are used for preparation of books, articles etc. All these versions and configurations must be kept well organized, otherwise you are bound to be lost. The Revision Control System (RCS) is an appropriate tool to assist with these tasks. With RCS it is possible, with small overhead, to preserve *all the revisions* which evolved from a given text document, to merge changes made by others, to compare different versions, and to keep a log of changes.

RCS

```
ci foo.tw                    (check-in latest version)
co foo.tw                    (check-out latest version)
co -r⟨rev⟩ foo.tw
rlog foo.tw
rcsdiff -r⟨rev⟩ foo.tw
rcsmerge -r⟨later_rev⟩ -r⟨earlier_rev⟩ foo.tw
```

When the first command is executed `foo.tw` is stored in a *group file* (with default name `foo.tw,v` on UNIX machines, or `foo.tw%` on MSDOS) as a new revision. For each revision you deposit, `ci` prompts for a log message. The file `foo.tw` is deleted unless you ask otherwise (`ci -l foo.tw`). The message "`ci error: no lock set by (login)`" tells you that RCS was configured with the 'strict locking feature' enabled. Locking prevents clashes between different users' modifications if several are working on the same file. This feature is disabled b the command `rcs -U foo.tw`; it is unnecessary if only the owner of the file is expected to deposit revisions into it.

The next two commands are used to extract the latest, or the specified, revision from the group file. `rlog` is used to print log messages. Different revisions of a document may be compared using `rcsdiff`. The command `rcsdiff foo.tw` compares the latest revision with the contents of the working file. The differences themselves are found by the program `diff`; if you do not like `diff`'s default output, change it by passing appropriate switches to `rcsdiff`. The last command undoes the changes between revisions; the file `foo.tw` will be overwritten. `rcsmerge` incorporates changes between two revisions into the working file. A similar effect could be achieved with a stand-alone program called `merge`. If files being compared are `mine`, `older`, `yours` then given the command

```
merge mine older yours
```

`merge` tries to add to `mine` the result of subtracting `older` from `yours`; if overlap occurs, i.e., both files `mine` and `yours` have changes to the same segment of lines in `older`, then `merge` delimits the alternatives with

```
<<<<<<< mine
(lines in) mine
=======
(lines in) yours
>>>>>>> yours
```

and writes above to `mine`. Now it is up to you which set of changes you adopt. `merge -p ...` sends the result of merging to the standard output.

To keep the working directory uncluttered, all RCS files are usually stored in the subdirectory with the name `RCS`. RCS commands look first into this directory when searching for files.

Concluding remarks

It seems that the TeX language constitutes a good starting point for exploring the idea of literate programming. The system is simple, because many features present in other WEBs are not needed. The system is extensible, which means that it is possible to try different styles and features. And finally, programs written in TeX are not too long — `plain.tex` is about 1000 lines of code — which means that you can print the documentation of real programs yourself and share it with others.

For those convinced by the analysis above, the literate source of `plain.tex` has been submitted to the CTAN archives, in directory `web/tweb`; please read it and enjoy.