Nuweb
A Simple Literate Programming Tool
version 0.90

Preston Briggs[1]
preston@cs.rice.edu
HTML scrap generator by John D. Ramsdell
ramsdell@mitre.org
Revised by Reuben Thomas
Reuben.Thomas@cl.cam.ac.uk

15th May 1997

[1]This work has been supported by ARPA, through ONR grant N00014-91-J-1989.

Contents

Chapter 1

Introduction

In 1984, Knuth introduced the idea of *literate programming* and described a pair of tools to support the practice [2]. His approach was to combine Pascal code with TeX documentation to produce a new language, `WEB`, that offered programmers a superior approach to programming. He wrote several programs in `WEB`, including `weave` and `tangle`, the programs used to support literate programming. The idea was that a programmer wrote one document, the web file, that combined documentation (written in TeX [3]) with code (written in Pascal).

Running `tangle` on the web file would produce a complete Pascal program, ready for compilation by an ordinary Pascal compiler. The primary function of `tangle` is to allow the programmer to present elements of the program in any desired order, regardless of the restrictions imposed by the programming language. Thus, the programmer is free to present his program in a top-down fashion, bottom-up fashion, or whatever seems best in terms of promoting understanding and maintenance.

Running `weave` on the web file would produce a TeX file, ready to be processed by TeX. The resulting document included a variety of automatically generated indices and cross-references that made it much easier to navigate the code. Additionally, all of the code sections were automatically pretty printed, resulting in a quite impressive document.

Knuth also wrote the programs for TeX and METAFONT entirely in `WEB`, eventually publishing them in book form [4, 5]. These are probably the largest programs ever published in a readable form.

Inspired by Knuth's example, many people have experimented with `WEB`. Some people have even built `WEB`-like tools for their own favorite combinations of programming language and typesetting language. For example, `CWEB`, Knuth's current system of choice, works with a combination of C (or C++) and TeX [7]. Another system, FunnelWeb, is independent of any programming language and only mildly dependent on TeX [9]. Inspired by the versatility of FunnelWeb and by the daunting size of its documentation, I decided to write my own, very simple, tool for literate programming.[1]

1.1 Nuweb

Nuweb works with any programming language and LaTeX 2_ε [6]. (Earlier versions of nuweb work with LaTeX2.09.) I wanted to use LaTeX because it supports a multi-level sectioning scheme and has facilities for drawing figures. I wanted to be able to work with arbitrary programming languages because my friends and I write programs in many languages (and sometimes combinations of several languages), such as C, Fortran, C++, yacc, lex, Scheme, assembly, Postscript, and so forth. The need to support arbitrary programming languages has many consequences:

[1]There is another system similar to mine, written by Norman Ramsey, called noweb [8]. It perhaps suffers from being overly Unix-dependent and requiring several programs to use. On the other hand, its command syntax is very nice. In any case, nuweb certainly owes its name and a number of features to his inspiration.

No pretty printing Both `WEB` and `CWEB` are able to pretty print the code sections of their documents because they understand the language well enough to parse it. Since we want to use *any* language, we've got to abandon this feature.

Manual indexing of identifiers Because `WEB` knows about Pascal, it is able to construct an index of all the identifiers occurring in the code sections (filtering out keywords and the standard type identifiers). Unfortunately, this isn't as easy in our case. We don't know what an identifier looks like in each language and we certainly don't know all the keywords. (On the other hand, see the end of section 1.2.)

Of course, we've got to have some compensation for our losses or the whole idea would be a waste. Here are the advantages I can see:

Simplicity The majority of the commands in `WEB` are concerned with control of the automatic pretty printing. Since we don't pretty print, many commands are eliminated. A further set of commands is subsumed by LaTeX and may also be eliminated. As a result, our set of commands is reduced to only four members. This simplicity is also reflected in the size of this tool, which is quite a bit smaller than the tools used with other approaches.

No pretty printing Everyone disagrees about how their code should look, so automatic formatting annoys many people. One approach is to provide ways to control the formatting. Our approach is simpler—we perform no automatic formatting and therefore allow the programmer complete control of code layout.

Control We also offer the programmer complete control of the layout of his program files (the files generated during tangling). Of course, this is essential for languages that are sensitive to layout; but it is also important in many practical situations, such as debugging.

Speed Since nuweb doesn't do too much, the nuweb tool runs quickly. I combine the functions of `tangle` and `weave` into a single program that performs both functions at once.

Page numbers Inspired by the example of noweb, nuweb refers to all scraps by page number to simplify navigation. If there are multiple scraps on a page (say page 17), they are distinguished by lower-case letters: 17a, 17b, and so forth).

Multiple file output The programmer may specify more than one program file in a single nuweb file. This is required when constructing programs in a combination of languages (say, Fortran and C). It's also an advantage when constructing very large programs that would require a lot of compile time.

A further reduction in compilation time is achieved by first writing each program file to a temporary location, then comparing the temporary file with the old version of the file. If there is no difference, the temporary file can be deleted. If the files differ, the old version is deleted and the temporary file renamed. This approach works well in combination with `make` (or similar tools), since `make` will avoid recompiling untouched program files. However, this does mean that you must run nuweb manually whenever you edit your web file (or run it automatically every time you use `make`). A solution is to produce a dummy program file, say `dummy`, and add a rule to the makefile like:

```
dummy: web.w
        rm dummy
        nuweb web.w
```

If `dummy` is made a dependent of `.INIT` or some other target that is made every time that `make` is run, then the web file will automatically be reprocessed whenever it is changed. The command `rm dummy` is needed to make sure that the file is rewritten every time the web is processed.

If you take a more modular approach, and only generate one program file from each web file, it is better to introduce an implicit rule for making program files from web files such as:

```
.w.c:
nuweb -tc $<
```

Note the use of the the `-t` flag to stop the documentation file being generated, and the `-c` flag (there's no point comparing the new version of the C file with the old, as we already know it needs to be remade).

1.1.1 Nuweb and HTML

In addition to producing LaTeX source, nuweb can be used to generate HyperText Markup Language (HTML), the markup language used by the World Wide Web. This can be done either from LaTeX or HTML source. With a LaTeX source, the tools which generate HTML automatically produce hypertext links. The remaining text describes the use of LaTeX markup; the use of HTML markup follows the same pattern.

1.2 Writing Nuweb

The bulk of a nuweb file will be ordinary LaTeX. In fact, any LaTeX file can serve as input to nuweb and will be simply copied through unchanged to the documentation file—unless a nuweb command is discovered. All nuweb commands begin with an at sign (`@`). Therefore, a file without at-signs will be copied unchanged. Nuweb commands are used to specify *program files*, define *macros*, and delimit *scraps*. These are the only features of interest to the nuweb tool—all else is simply text to be copied to the documentation file.

1.2.1 The Major Commands

Files and macros are defined with the following commands:

`@o` ***file-name flags scrap*** Output a program file. The file name is terminated by whitespace.

`@d` ***macro-name scrap*** Define a macro. The macro name is terminated by a return or the beginning of a scrap.

A specific file may be specified several times, with the definitions being written out, one after another, in the order they appear. The definitions of macros may be similarly divided.

Macro names may be (almost) any well-formed TeX string. Changing fonts is allowed, as is using math mode; however, since in the `alltt` environment used to display code the $ sign is an ordinary character, the commands `\(` and `\)` should be used instead to enter and leave math mode. Also, spaces are significant, so avoid constructs such as `{\tt xyz}`; use `\texttt{xyz}` instead. `\verb` cannot be used in macro names; `\tt` or `\texttt` should be used instead. When producing HTML, macros are displayed in a preformatted element (`PRE`), so macros may contain one or more `A`, `B`, `I`, `U`, or `P` elements or data characters.

Very long scraps may be allowed to break across a page if declared with `@O` or `@D` (instead of `@o` and `@d`). This doesn't work very well as a default, since far too many short scraps will be broken across pages; however, as a user-controlled option, it seems very useful. No distinction is made between the upper case and lower case forms of these commands when generating HTML.

Scraps

Scraps have specific start and end markers to allow precise control over the contents and layout. Note that any amount of whitespace (including carriage returns) may appear between a name and the beginning of a scrap. A scrap has the form:

`@{`***anything***`@}` where the scrap body includes every character in *anything*—all the blanks, all the tabs, all the carriage returns.

Inside a scrap, we may invoke a macro:

`@<`***macro-name***`@>` Causes the macro *macro-name* to be expanded inline as the code is written out to a file. Recursive macro invocations are not allowed.

Note that macro names may be abbreviated, either during invocation or definition. For example, it would be tedious to have to type the macro name

```
@d Check for terminating at-sequence and return name if found
```

repeatedly. Therefore, we provide a mechanism (borrowed from Knuth) of indicating abbreviated names.

```
@d Check for terminating...
```

The programmer need only type enough characters to uniquely identify the macro name, followed by three periods. An abbreviation may even occur before the full version; nuweb simply preserves the longest version of a macro name. Note also that strings of blanks and tabs are replaced by a single blank.

When scraps are written to a program file or documentation file, tabs are expanded into spaces by default. Currently, I assume tab stops are set every eight characters. Furthermore, when a macro is expanded in a scrap, the body of the macro is indented to match the indentation of the macro invocation. Therefore, care must be taken with languages such as Fortran that are sensitive to indentation. These default behaviors may be changed for each program file (see below).

Flags

When defining a program file, various flags may be used to control its appearance. The flags are intended to make life a little easier for programmers using certain languages. They introduce little language dependencies; however, they do so only for a particular file. Thus it is still easy to mix languages within a single document. There are three per-file flags:

`-d` Forces the creation of `#line` directives in the program file. These are useful with C (and sometimes C++ and Fortran) on many systems since they cause the compiler's error messages to refer to the web file rather than the program file. Similarly, they allow source debugging in terms of the web file.

`-i` Suppresses the indentation of macros. That is, when a macro is expanded in a scrap, it will *not* be indented to match the indentation of the macro invocation. This flag seems most useful for Fortran programmers.

`-t` Suppresses expansion of tabs in the program file. This feature is useful when generating `make` files.

1.2.2 The Minor Commands

There are two very low-level utility commands that may appear anywhere in the web file.

`@@` Causes a single at sign to be copied into the program.

`@i` ***file-name*** Includes a file. Includes may be nested, though there is currently a limit of 16 levels. The file name should be complete (no extension will be appended) and should be terminated by a newline.

1.2.3 Indices

Finally, there are three commands used to create indices to the macro names, file definitions, and user-specified identifiers.

`@f` Create an index of file names.

`@m` Create an index of macro names.

`@u` Create an index of user-specified identifiers.

These commands produce a series of `\item` commands in the LaTeX file, and so must be placed within a list environment of some sort. If there is only one index, then the `theindex` environment is suitable; otherwise, the `nuweb` package (see section 1.2.4) supplies the `nuwebindex` environment. A typical index will look like this:

```
\begin{nuwebindex}[2]{\section{Index}}
@u
\end{nuwebindex}
```

The optional argument to the `nuwebindex` environment gives the number of columns to use, which must be from one to ten. The default is two, which is usually appropriate, but one is useful for the macros index. The second argument is text which will be put in a single column just before the index, and kept on the same page. This prevents a problem where having a `\section` which starts at the top of a page can cause a page break before the `nuwebindex` environment, because when the `\begin{nuwebindex}` is executed the `\section` has not yet been pushed to the top of the following page, so a page break is inserted after it, even though later it will turn out to be unnecessary.

Index entries are not broken over a column or page; this prevents problems as nuweb generates long index lists for frequently used identifiers, which can span several lines. Unlike the `theindex` environment, no new chapter is started; this can be done manually if desired, using a command such as `\chapter*{Index}`.

Identifiers must be explicitly specified for inclusion in the `@u` index. By convention, each identifier is marked at the point of its definition; all references to each identifier inside scraps will be discovered automatically. To mark an identifier for inclusion in the index, it is included after the sign `@|` at the end of a scrap. For example,

```
    @d a scrap
    @{ Let's pretend we're declaring the variables
    FOO and BAR inside this scrap.
    @| FOO BAR
    @}
```

I've used alphabetic identifiers in this example, but any string of characters (not including white-space or `@` characters) will do. Therefore, it's possible to add index entries for things like `<<=` if desired. An identifier may be declared in more than one scrap.

In the generated index, each identifier appears with a list of all the scraps using and defining it, where the defining scraps are distinguished by underlining. If the `-n` flag is used to nuweb (see section 1.3), these scrap numbers will not be page numbers. Note that the identifier doesn't actually have to appear in the defining scrap; it just has to be in the list of definitions at the end of a scrap.

1.2.4 The **nuweb** package

When nuweb is used to produce LaTeX output, the `nuweb` package must be used, as it provides commands that nuweb uses to perform its formatting. If you want to change the way scraps are formatted, you can redefine these commands; for details, see chapter 4 of the full documentation (produced from `nuweb.tex`).

Two facilities are provided for the user: the `nuwebindex` environment (see section 1.2.3), and commands to add a terminating symbol to the end of each scrap; some authors feel this makes webs easier to read. The commands are:

> `\nuwebterminator{`*command*`}`

where *command* is executed at the end of every scrap. In earlier versions of nuweb, a diamond ($\Diamond$) was used; to obtain this effect, use the command

```
\nuwebterminator{\(\Diamond\)}
```

To have no terminating symbol (the default), use the command

```
\nuwebnoterminator
```

1.3 Running Nuweb

Nuweb is invoked using the following command:

> `nuweb` *flags [*`-o` *file] file...*

One or more files may be processed at a time; each file may have the name of its documentation file specified, using `-o` *file*. The documentation file name must include the extension. If a file name has no extension, `.w` will be appended. LaTeX suitable for translation into HTML by LaTeX2HTML will be produced from files whose name ends with `.hw`; source marked up in HTML should have the extension `.htw`. While a file name may specify a file in another directory, the resulting documentation file will always be created in the current directory unless the output filename is specified using the `-o` flag. For example,

```
nuweb /foo/bar/quux
```

will take as input the file `/foo/bar/quux.w` and will create the file `quux.tex` in the current directory.

By default, nuweb performs both tangling and weaving at the same time. Normally, this is not a bottleneck in the compilation process; however, it's possible to achieve slightly faster throughput by avoiding one or another of the default functions using command-line flags. There are currently five possible flags:

`-o` ***file*** Call the documentation file *file*. The filename must include the extension.

`-t` Suppress generation of the documentation file.

`-p` Suppress generation of the program files.

`-c` Avoid testing program files for change before updating them.

`-C` Ignore `#line` directives when testing program files for change. This avoids source files being regenerated just because you have changed the line numbers at which their code appears in the nuweb document. Using this option will save a lot of remaking, but may result in debug information being out of date. Compiler error messages should always be correct, though.

Thus, the command

```
nuweb -to /foo/bar/quux
```

would simply scan the input and produce no program at all.

There are three additional command-line flags:

`-v` For "verbose", causes nuweb to write information about its progress to `stderr`.

`-n` Forces scraps to be numbered sequentially from 1 (instead of using page numbers). This form is perhaps more desirable for small webs. This option applies only when scraps are generated in LaTeX format.

`-s` Normally, if `@{` is followed by a newline, the newline is ignored when formatting the scrap, and similarly with a newline before `@}` or `@|`. This allows source code to start on the line after `@{` and end on the line before `@}` or `@|`, which is convenient for editing, without leaving blank lines in the LaTeX file. Using this flag, the scraps are formatted verbatim; scraps are always written out to program files verbatim.

It is worth noting that source code often contains wide lines, so when generating LaTeX documents it may help to use a package such as a4wide to avoid getting too many "`overfull hbox`" messages.

1.4 Generating HTML

Nikos Drakos's LaTeX2HTML [1] can be used to translate LaTeX with embedded HTML scraps into HTML. Be sure to use the package html so that LaTeX will understand the hypertext commands. When translating into HTML, do not allow a document to be split by specifying "`-split 0`". You need not generate navigation links, so also specify "`-no_navigation`".

While preparing a web, you may want to view the program's scraps without taking the time to run LaTeX2HTML. Simply rename the generated `.tex` file so that its file name ends with `.html`, and view that file. The documentation section will be jumbled, but the scraps will be clear.

1.5 Restrictions

Because nuweb is intended to be a simple tool, I've established a few restrictions. Over time, some of these may be eliminated; others seem fundamental.

- File names and indexed identifiers must not contain any `@` signs. This may be changed in future.
- Anything is allowed in the body of a scrap; however, very large scraps (horizontally or vertically) do not typeset well.
- Temporary files (created for comparison with the existing program files) are placed in the current directory. Since they may be renamed to an program file name, all the program files must be on the same file system as the current directory.
- Because page numbers cannot be determined until the document has been typeset, we have to rerun nuweb after LaTeX to obtain a clean version of the document (very similar to the way we sometimes have to rerun LaTeX to obtain an up-to-date table of contents after significant edits). Nuweb will warn (in most cases) when this needs to be done; in the remaining cases, LaTeX will warn that labels may have changed.
- Tabs are expanded to eight spaces; a per-file flag might in future be added to specify how many spaces should replace a tab.
- Nuweb is still not completely portable: it expects file-names to use / to delimit directories.

1.6 Acknowledgements

Several people have contributed their times, ideas, and debugging skills. In particular, I'd like to acknowledge the contributions of Osman Buyukisik, Manuel Carriba, Adrian Clarke, Tim Harvey, Michael Lewis, Walter Ravenek, Rob Shillingsburg, Kayvan Sylvan, Dominique de Waleffe, and Scott Warren. Of course, most of these people would never have heard of nuweb (or many other tools) without the efforts of George Greenwade.

Preston Briggs

Preston Briggs and John Ramsdell encouraged and advised me when I suggested my revisions, and kindly permitted me to distribute the result.

Reuben Thomas

Bibliography

[1] Nikos Drakos. The LaTeX2html translator, 1994. Program available from CTAN nodes in `/ctan/tex-archive/support/latex2html/`.

[2] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, May 1984.

[3] Donald E. Knuth. *The TeXbook*. Computers & Typesetting. Addison-Wesley, 1986.

[4] Donald E. Knuth. *TeX: The Program*. Computers & Typesetting. Addison-Wesley, 1986.

[5] Donald E. Knuth. *METAFONT: The Program*. Computers & Typesetting. Addison-Wesley, 1986.

[6] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 1986.

[7] Silvio Levy and Donald E. Knuth. `CWEB` user manual: The `CWEB` system of structured documentation. Technical Report STAN-CS-83-977, Stanford University, October 1990. Available for anonymous ftp from `labrea.stanford.edu` in directory `pub/cweb`.

[8] Norman Ramsey. Literate-programming tools need not be complex. Submitted to IEEE Software, August 1992.

[9] Ross N. Williams. FunnelWeb user's manual, May 1992. Available for anonymous ftp from `sirius.itd.adelaide.edu.au` in directory `pub/funnelweb`.