FWEB

A WEB system of structured documentation
for multiple languages

John A. Krommes

FWEB Copying Permissions

1 INTRODUCTION to FWEB

FWEB is a system for "literate programming". It was originally intended for scientific programming, and is in wide use in that arena; however, it has much broader applicability. It is an extension of Knuth's WEB system that handles the specific languages C, C++, Fortran, Ratfor, and TEX. It also supports a WYSIWYG language-independent mode as well as a (closely-related but not identical) verbatim 'language'.

The origins of literate programming are described in the very enjoyable book by D. E. Knuth, "Literate Programming" (Center for the Study of Language and Information, Leland Standard Junior University, 1992).

Knuth's original WEB was written in Pascal and formatted Pascal code. Silvio Levy introduced CWEB, a WEB system written in C for C. FWEB is a (by now, substantial) modification of an early version of CWEB that was graciously supplied by Levy. It also borrows various ideas from the work of Ramsey and Briggs on language-independent webs.

1.1 History of WEB and literate programming

(To be completed; see Knuth's book.)

1.2 FWEB features

FWEB is distinguished from its relatives in several respects:

- FWEB introduces the concept of a current language (see Chapter 8 [Languages], page 62), so more than one compiler language can be processed in a single WEB run.
- FWEB understands the syntaxes of several of the more important compiler languages: C, C++, Fortran, Ratfor, and TEX. For other languages, FWEB can work in a language-independent mode that essentially weaves and tangles the source code verbatim, but still provides the user with the powerful WEB features related to TEX documentation, module names, macro processing, etc.
- FWEB contains a built-in Ratfor (RATional FORtran) translator. See Chapter 9 [Ratfor], page 65.

- FWEB has a built-in C-like macro preprocessor, which should be especially useful for Fortran and Ratfor. See Chapter 7 [Macros], page 48, Section 7.3 [Preprocessing], page 59.
- Many aspects of FWEB's behavior can be customized by means of setting parameters in a style file.

2 WEB CONCEPTS

The principle concepts of WEB programming are laid out in Knuth's book, the reference to which is given in Chapter 1 [Intro], page 2. FWEB follows most conventions introduced by WEB and CWEB, except that the names of some commands have been changed for consistency, symmetry, and/or clarity.

2.1 The FWEB processors

Following Knuth's original design, FWEB consists of two processors, FTANGLE and FWEAVE. Both operate on a single source file, say '`test.web`'. FTANGLE produces compilable code, say '`test.c`', whereas FWEAVE produces a TeX file, '`test.tex`', that can be processed with TeX or LaTeX. For detailed descriptions of the LaTeX support, see Section 10.1.2 [LaTeX], page 68.

2.2 The structure of a web

An FWEB source file is structured into sections. Each section consists of three (optional) parts: the

1. TeX part;
2. definition part; and
3. code part.

Sections can be combined into larger units called *modules*, as explained in Section 2.3 [Modules], page 6.

A simple example of an FWEB source file consisting of two sections is as follows:

```
@n

@* INTRODUCTION.
This code is intended...

@m PI 3.14159

@a
```

```
          program main
          call compute
          end

@ The computational routine is pretty boring.
@a
          subroutine compute
          write(*,*) 'pi = ', PI
          end
```

Commands to FWEB are begun by the '@' symbol (see Chapter 5 [AT commands], page 29). In this example, the first command, '@n', sets the global language to Fortran-77. For more information about languages, see Chapter 8 [Languages], page 62.

The '@*' command begins a major section (corresponding to LaTEX's `\section` command); this command is followed by the section name, terminated by a period. Major sections are entered in the table of contents. The command '@*n' begins a major (sub)section of level n, where '@*0' is equivalent to the simple '@*', '@*1' indicates a subsection, '@*2' indicates a subsubsection, and so on.

Minor sections are begun by '@ '; these have no associated names and are not entered into the table of contents.

All sections begin with TEX commentary. This is followed by an optional definition part. The beginning of the definition part is signaled by the appearance of any one of the commands '@d', '@f', '@m', '@v', '@W'. In the example, the first section has a definition part consisting of one macro definition ('@m'); the second section has no definition part.

An unnamed code part is begun by '@a'. A named code part is begun by the appearance of a module name such as '@<Global variables@>', followed by an equals sign; see Section 2.3 [Modules], page 6.

The portion of the source file before the first section (i.e., before the first '@*' or '@ ') is called 'in limbo' or 'the limbo part'. The only '@' commands that are allowed in limbo (in addition to '@@', which is allowed anywhere) are the language-changing commands, and one of those should appear. Other text in limbo is ignored by FTANGLE and is copied by FWEAVE to the output file. Thus, one can make TEX macro definitions in limbo that override the defaults in FWEB's style file 'fwebmac.sty'. (Another way of getting TEX text into the limbo part is by means of the '@l' command; see Section 5.5.10 [ATl], page 34.)

2.3 Modules

Sections can be combined into modules. Modules can be named or unnamed. There is just one unnamed module.

2.3.1 The unnamed module

The unnamed module is introduced by the command '`@a`'. Subsequent uses of '`@a`' accrete code to the unnamed module. FTANGLE *outputs the unnamed module.*

2.3.2 Named modules

A module name is specified by

```
@< Arbitrary TeX text @>
```

Leading and trailing white space around the name text is ignored.

To define a named module, replace the '`@a`' that begins the unnamed code part of a section by '`@< module name @>=`'. If one uses this construction with the same name in a later section, the effect is to *accrete* to the contents of the original section. Thus, a named module might ultimately consists of the code from sections 2, 5, and 8, for example.

To use a named module, simply use the name anywhere in a code part; FTANGLE will insert the contents of the module at the point where the name is used. For example,

```
@c
@ Here's how to use a named module.
@a
for(i=1; i<n; i++)
        @< Inner loop @>@;

@ Here's how to define a named module.  Definitions may occur after use.
@< Inner...@>=
{
a[i] = i;
}
```

After a name has appeared once in full, it may be abbreviated by a unique prefix followed by 3 dots, as demonstrated in the above example. By convention, a complete module name cannot be a subset of another. For example, ‘`@<Test@>`’ and ‘`@<Test of graphics@>`’ will elicit an error message.

Commonly the first unnamed section in the code indicates its modular structure. For example, a C code might begin with

```
@c
@* DEMO.
@a
@<Include files@>@;
@<Typedefs@>@;
@<Function prototypes@>@;
@<Global variables@>@;
```

This way, global variable definitions can be introduced as needed, but will be guaranteed to appear at the top of the code. Function prototypes could be handled this way as well; alternatively, they could all be collected into one section, perhaps at the end of the source file. (The above organization still guarantees that they will appear at the beginning of the output.) Functions could be introduced one at a time in unnamed sections.

2.4 Phases of processing

The FWEB processors perform their work in several distinct phases.

2.4.1 The phases of FTANGLE

FTANGLE has two phases. In phase 1, the source file is read; in phase 2, compilable code is written out in the order specified by the web.

More specifically, phase 1

- discards TeX documentation;
- tokenizes source;
- expands ‘`@'...'`’, ‘`@"..."`’, and ‘`0b...`’ (in Fortran, also ‘`0...`’ and ‘`0x...`’);
- stores code text in appropriate modules;

- memorizes macro definitions (‘@d’ and ‘@m’). See Section 5.5.5 [ATd], page 32, See Section 5.5.12 [ATm], page 35.

Phase 2

- outputs outer macro definitions (‘@d’);
- outputs the unnamed module (‘@a’);
- expands WEB macros (‘@m’);
- expands built-in macros (see Section 7.2.3 [Built-in functions], page 50);
- translates Ratfor statements. See Chapter 9 [Ratfor], page 65

2.4.2 The phases of FWEAVE

FWEAVE has three phases. In phase 1, the source file is read and cross-reference information is collected. In phase2, the source file is read again, then pretty-printed including the cross-reference information. In phase 3, the index and table of contents are written.

More specifically, phase 1

- tokenizes and stores identifiers and module names;
- collects cross-reference information;
- stores limbo text definitions (‘@l’) (see Section 5.5.10 [ATl], page 34);
- collects information about overloaded operators (‘@v’) and identifiers (‘@W’). See Section 5.5.21 [ATv], page 37, Section 5.5.22 [ATW_], page 37.

Phase 2

- outputs limbo text;
- outputs special TeX macros for overloaded operators;
- copies TeX material directly to output;
- treats material between vertical bars (‘|...|’) as code to be typeset;
- tokenizes and stores contents of each code section;
- analyzes code syntax and converts to appropriate TeX macros.

3 FILES

FWEB works with a variety of files. File names have the form '`[path]/root[.ext]`', where the brackets denote optional. Here the slash is called the prefix end character. Since this character differs for various operating systems, it can be changed in '`custom.h`' (see Chapter 11 [Customization], page 74). The character that initiates the file-name extension (normally a period) can be changed with the '`-E`' command-line option (see Section 4.2.12 [-E_], page 15).

3.1 Input files

'`.fweb`' — Initialization file; always in the home directory.

'`fweb.sty`' — Style file; in the directory of the WEB file unless changed by environment variable `FWEB_STYLE_DIR`. The basic name can be changed by the '`-z`' option (see Section 4.2.61 [-z], page 26).

'`name.web`' — Source code.

'`name.ch`' — Change file. See Section 3.3 [Change files], page 11.

'`name.hweb`' — Code included into web file with '`@i`' (see Section 5.5.7 [ATi], page 33). Include files are searched for in the path set by the environment variable `FWEB_INCLUDES` and/or the '`-I`' option (see Section 4.2.17 [-I_], page 16). If that path is empty, then the current directory is searched.

'`name.hch`' — Change file for include file.

3.1.1 Automatic file-name completion

Automatic completion of input file names is turned on by the '`-e`' command-line option (see Section 4.2.13 [-e], page 15). When in effect, input file names that include no period (have no extension) are completed automatically according to the contents of the following style-file entries:

Type of file	Style-file entry	Default
WEB file	'`ext.web`'	'`.web`'

```
Change file        'ext.ch'          '.ch'
Include file       'ext.hweb'        '.hweb'
Change file
 for include file  'ext.hch'         '.hch'
```

More than one extension may be specified, as a space-delimited list—e.g., '"web wb"'; the first one that matches is used.

3.2 Output files

'name.tex' — Woven output to be processed with TeX or LaTeX.

'CONTENTS.tex' — Temporary file that accumulates table-of-contents information.

'INDEX.tex' — Temporary file that stores indexing information.

'MODULES.tex' — Temporary files that stores module list.

'name.ext' — Compilable output file.

The names of the three temporary files can be changed with style-file options. See Section 11.3 [Style], page 75. Commonly, one may put into the style file 'fweb.sty' commands such as

```
index.tex "#.ndx"
modules.tex "#.mds"
contents.tex "#.cdts"
```

The '#' is replaced by the root name of the web file.

The extensions for the compilable output file(s) have certain defaults, but can be changed by style-file parameters according to the following table:

Language	Style-file entry	UNIX default	non-UNIX default
C	suffix.C	.c	.c
C++	suffix.Cpp	.c++	.cpp
Fortran--77	suffix.N	.f	.for
Fortran--90	suffix.N90	.f	.for
Ratfor--77	suffix.R	.f	.for

`Ratfor--90`	`suffix.R90`	`.f`	`.for`
`TeX`	`suffix.X`	`.sty`	`.sty`
`VERBATIM`	`suffix.V`	`.mk`	`.mk`

3.3 Change files

The primary input to the FWEB processors is the '`test.web`' source. However, a *change file* '`test.ch`' can also be specified. A change file consists of instances of the following structure:

```
@x
(One or more lines of text, EXACTLY as in the web file.  Copy these
lines with the editor; don't type them from scratch.)
@y
(Replacement text.)
@z
```

The change-file mechanism allows one to insert local changes or test new code without physically modifying the original web file.

To specify a change file, use its name as the second file name on the command line. The extension '`.ch`' is assumed by default. For example,

FTANGLE `test test`

processes '`test.web`' with the change file '`test.ch`'.

In addition to '`@x`', '`@y`', and '`@z`', the only '`@`' commands allowed in a change file are language-changing commands such as '`@c`' and the special commands '`@[`' and '`@]`'. The command '`@[`' is used for column-oriented languages such as Fortran–77 and means *switch into code mode*. Similarly, '`@]`' means *switch out of code mode*.

All '`@`' commands in a change file must begin in column 1. Lines not beginning with '`@`' are ignored, so may be used as comments.

4 RUNNING FWEB

FWEB has a Unix-style command-line syntax. There are many command-line options, but most of these are unnecessary for standard appplications.

Commonly-used command-line options can be placed into the initialization file '.fweb'. See Section 4.2 [Options], page 12.

4.1 Command-line syntax

The command-line syntax is

```
ftangle-|
        | webfile[.web] [changefile[.ch]] [-option...]
fweave--|
```

File names are anything that doesn't begin with a '-'. Options begin with a '-'. File names and options can be intermixed, or the options may appear before the file names. The first file name encountered is the web file (source file); the second, if it exists, is the change file. [When no change file is specified, FWEB attempts to read from the null file ('/dev/null' on Unix systems). This name should be specified when FWEB is installed, or can be set in the style file. See Section 11.3.6.13 [null_file], page 80.]

An option consisting of a lone hyphen stands for the special file name 'stdin', which means 'read from the standard input.' (This option should not be used except for very special effects.)

The web file is shown as required since one is normally processing a source. However, a few of the information options (see Section 10.2.4 [Info options], page 73) will work even in the absence of a web file.

4.2 Command-line options

Command-line options may be put into the initialization file '.fweb' (which is always in the user's home directory). In that file, options beginning with a plus sign are processed before the command-line options. Otherwise, they are processed after the command-line options.

4.2.1 Negating options

To negate a command-line option, use an extra hyphen. For example, '--v' means 'Don't make all comments verbatim.' This kind of construction is useful if an option such as '-v' is turned on in the '.fweb' initialization file and you wish to turn it off for just one run.

4.2.2 '-1': Turn on brief debugging mode (FWEAVE).

This option tells FWEAVE to display irreducible scraps. (Need discussion somewhere of scraps.)

4.2.3 '-2': Turn on verbose debugging mode (FWEAVE).

This commands tells FWEAVE to display detailed reductions of the scraps as it does the pretty-printing. Sometimes FWEAVE fails spectacularly at printing-printing a section, either because of a syntax error on the part of the user or because of a bug in FWEAVE's logic. This option helps one to figure out why.

This feature can be turned on more locally by means of the '@2' command. See Section 5.1.3 [AT2], page 29.

4.2.4 '-@': Display the control-code mappings.

This command supplies information about the '@' control codes: it shows the associated style-file parameters that can be used to remap the codes, and it displays the precedence. (Some codes may be used anywhere; others begin a new part. Codes that begin the definition part are labelled by [D]; codes that begin the code part are labelled by [C]; codes that begin a new section are labelled by [S].)

4.2.5 '-A_': Turn on ASCII translations.

This command is used primarily for debugging. FWEB works internally with the ASCII character set. If FWEB is run on a non-ASCII machine, translations to and from the internal ASCII are done automatically; on an ASCII machine, these translations are unnecessary and are not performed unless the '-A' option is used.

4.2.6 '-B': Turn off audible beeps.

FWEB sometimes beeps the terminal for certain errors. The '-B' option turns off the beeps, replacing them by a printed exclamation point.

(This option is sometimes called the "marriage-saver", after the situation that prompted a user's request for this feature.)

4.2.7 '-b': Number blocks.

Number **do** and **if** blocks in woven Fortran and Ratfor output.

4.2.8 '-c': Set global language to C.

See Chapter 8 [Languages], page 62, Section 8.2 [C], page 63.

4.2.9 '-c++': Set global language to C++.

See Chapter 8 [Languages], page 62, Section 8.2.2 [Cpp], page 64.

4.2.10 '-D': Display reserved words.

This option displays the list of reserved words for the language currently in force. Thus, to see the reserved words for Ratfor–90, say

```
ftangle -Lr9 -D
```

For this option one must set the language on the command line, because the '-D' option is processed before the limbo section of the web file is read.)

If you say '-Dabc', you will get just the reserved words that begin with "abc".

4.2.11 '-d': Convert do...enddo.

Obsolete.

4.2.12 '-E': Change the delimiter of a file-name extension.

The standard delimiter for file-name extensions is a period, as in 'test.web'. To change this character to a comma, for example, say '-E,'.

4.2.13 '-e': Turn on automatic file-name completion.

When the '-e' option is in effect, FWEB attempts to be helpful in figuring out what file name you intend. For any input file name that has no extension (no embedded period), FWEB completes the name by adding the extension contained in the style-file parameter listed in the following table:

```
Type of file    Style-file entry      Default

WEB file          'ext.web'           '.web'
Change file       'ext.ch'            '.ch'
Include file      'ext.hweb'          '.hweb'
Change file
 for include file  'ext.hch'           '.hch'
```

More than one extension may be specified, as a space-delimited list—e.g., '"web wb"'; the first one that matches is used.

4.2.14 '-F': Compare output files with old versions.

When the '-F' option is in effect, FTANGLE writes its output to a temporary file (or files) instead of to its ultimate destination—e.g., 'test.c' and/or 'test.f'. After all output is written, the temporary files are compared with the old version of the files, if they exist. If the files are identical, the appropriate temporary file is deleted; otherwise, the temporary file is renamed, effectively overwriting the old version. This feature avoids updating the time stamp on the file unnecessarily, so 'make' files won't recompile the output unless it really has to.

(Note that with this option in effect, if one used the Unix utility 'touch' to force processing of a group of files, but the WEB sources are never changed, the 'make' file will continue to tangle the sources no matter how many times it is run, since FTANGLE will never update the time stamp on the files.)

The location of the temporary file as well as details of the renaming procedure are determined by the automatic configuration script during installation of the processors. The script `./configure` first looks for the (non-ANSI) function `tempnam`. If it finds it, it uses it to place the temporary file in the directory that FWEB would normally use for output in the absence of the '`-F`' option. (That is usually the current directory.) If `tempnam` is not available, the ANSI routine `tmpnam` is used. That places the temporary file in a directory determined by the system.

To implement the renaming, the `rename` function is used. This may fail if `tmpnam` placed the temporary file on a different device. If so, an attempt is made to force the rename by using the |system| routine to issue a `mv` command. Terminal output indicates the progress of the renaming. An asterisk following an output file name indicates that `rename` did not succeed, but the `mv` command did.

4.2.15 '`-f`': Turn off module references for identifiers.

By default, FWEAVE gives certain entities such as function names a section-number subscript to indicate where that function was defined. The '`-f`' command turns off the subscripting operation.

The subscript operations are further controlled by the settings of the style-file parameters `mark_defined`. See ⟨undefined⟩ [S_mark_defined], page ⟨undefined⟩.

4.2.16 '`-h`': Get help.

This just prints a message saying where further help is available.

4.2.17 '`-I`': Append to search list for include files.

The fundamental search list for include files is defined by the environment variable `FWEB_INCLUDES`, which is a colon-delimited list such as

```
setenv FWEB_INCLUDES .:/usr/fweb:/other/stuff
```

The '`-I`' option appends to this list.

4.2.18 '-i': Don't print '@I' include files.

If a web file is included via '@I' (see Section 5.5.8 [ATI_], page 34), for example

```
@I formats.hweb
```

then the '-i' option means to read and process the web file, but don't print its contents. This option is often used for large files of macro definitions, formats, or typedef statements that must be included at the beginning of even very short web files; it clutters things up to print such header files all the time.

Note that files included via '@i' do not respond to '-i' or '-i!'.

4.2.19 '-i!': Don't read '@I' include files.

If a web file is included via '@I', for example

```
@I formats.hweb
```

then the '-i!' option means to ignore such files completely. This option is seldom useful; the '-i' option (see Section 4.2.18 [-i], page 17) is more often used.

4.2.20 '-L': Select global language.

To select a global language from the command line, say '-Ll', where l is one of {c,c++,n,n9,r,r9,x}. See Chapter 8 [Languages], page 62.

4.2.21 '-l': Echo input line.

The option '-l[mmm[:nnn]]' echoes the input line constructed by the input driver between lines *mmm* and *nnn*. Missing *nnn* means echo to the end of file. Missing *mmm* means echo from the beginning.

This option is useful as a debugging tool. It is often used to verify that the input driver is inserting semicolons correctly. For Fortran–77, it is also useful to verify that comments are being processed correctly.

4.2.22 '-m': Define WEB macro.

The command-line construction

```
-mA(x)=x
```

defines the WEB macro A as though the definition

```
@m A(x) x
```

had appeared at the beginning of the web file.

4.2.23 '-m4': Understand m4 built-in commands.

This tells FWEAVE to properly format the reserved words of the m4 preprocessor. The use of this preprocessor is *not recommended*; use FWEB's built-in preprocessor instead.

4.2.24 '-m;': Append pseudo-semicolons.

When '-m;' is in effect, the construction '@;' is appended automatically to all WEB macro definitions.

This option is not recommended. Insert the '@;' by hand when necessary, as in

```
@m SET(x,y) x=1; y=2@;
```

4.2.25 '-n': Set global language to Fortran–77.

See Chapter 8 [Languages], page 62, Section 8.2.3 [Fortran], page 64.

4.2.26 '-n9': Set global language to Fortran–90.

See Chapter 8 [Languages], page 62, Section 8.2.3 [Fortran], page 64.

4.2.27 '-n;': Supply automatic semicolons (Fortran).

This is the default mode of operation in Fortran–77; the input driver automatically appends a semicolon to each logical line of source code. Since it's the default, you don't have to use it unless you wish to negate it. See Section 4.2.1 [Negating options], page 13.

4.2.28 '-nb': Number ifs and dos (Fortran).

In the woven output, extra comments are added to help one correlate the block structure of the code.

4.2.29 '-np': Print semicolons (Fortran).

Although the Fortran input driver automatically terminates logical lines with semicolons so that the innards of FWEAVE can process them correctly, the semicolons are by default not printed. To make them be printed, use the '-np' option.

4.2.30 '-n\': Free-form syntax continued by backslash.

In Fortran–90, this turns on free-form syntax and sets the continuation character to be the backslash. For example,

```
-n9[-n\]
@
@a
program main;
x = \
 y;
end;
```

In the tangled output the backslash is converted into Fortran-90's standard continuation character, the ampersand.

4.2.31 '-n&': Free-form syntax continued by ampersand.

In Fortran–90, this turns on free-form syntax and sets the continuation character to be the backslash. For example,

```
-n9[-n\]
@
@a
program main;
x = &
 y;
end;
```

4.2.32 '-n/': Recognize short comments (Fortran).

The standard FWEB notation for a short comment is '// ...'. However, in Fortran the '//' denotes concatenation by default. To make it denote a short comment, use the '-n/' option. For concatenation, use '\/'.

4.2.33 '-n!': Make '!' denote short comment (Fortran).

4.2.34 '-n)': Reverse array indices (Fortran)

This flag permits Fortran programmers to use C-style array indices. Conversions such as the following are made:

```
a(k)(i) => a(i,k)
a(k)(i,j) => a(i,j,k)
a(k)(j)(i) => a(i,j,k)
```

This feature permits convenient definitions of macros to deal with multi-dimensional vectors.

4.2.35 '-o': Don't overload operators.

(To be finished.)

4.2.36 '-q': Don't translate Ratfor.

(Obsolete.)

4.2.37 '-P': Select TeX processor.

Say '-PT' or '-PL' to inform FWEAVE that its output will be processed by TeX or LaTeX, respectively. If you always use LaTeX, it's easiest to put '+PL' into the '.fweb' initialization file.

4.2.38 '-p': Buffer up a style-file entry.

This option can be used in the '.fweb' initialization file to record style-file entries that are common to all runs. This command buffers up the entries. They are effectively placed at the beginning of the local style file, which is read after the command line is processed. See Section 11.3 [Style], page 75.

4.2.39 '-r': Set global language to Ratfor–77.

See Chapter 8 [Languages], page 62, Chapter 9 [Ratfor], page 65.

4.2.40 '-r9': Set global language to Ratfor–90.

See Chapter 8 [Languages], page 62, Chapter 9 [Ratfor], page 65.

4.2.41 '-rg': Set goto parameters.

This obscure parameter is used for Ratfor (and really should be a style-file parameter). Please see the manual.

4.2.42 '-k': Suppress comments about Ratfor translation.

By default, the Ratfor translator writes comments about what command it is translating. The '-k' option suppresses those comments. Arguments to this command allows one to suppress comments about only particular commands; please see the reference guide.

4.2.43 '-K': Write comments about Ratfor translation.

This is the negative of '-k'; it forces comments about particular Ratfor commands.

4.2.44 '-r;': Turn on auto-semi mode (Ratfor).

Please don't use this command. Insert semicolons by hand in your Ratfor code.

4.2.45 '-rb': Number `ifs` and `dos` (Ratfor).

In the woven output, extra comments are added to help one correlate the block structure of the code.

4.2.46 '-r/': Recognize short comments (Ratfor).

The standard FWEB notation for a short comment is '`// ...`'. However, in Ratfor the '`//`' denotes concatenation by default. To make it denote a short comment, use the '`-r/`' option. For concatenation, use '`\/`'.

4.2.47 '-r!': Make '!' denote short comment (Ratfor).

4.2.48 '-r)': Reverse array indices (Ratfor)

4.2.49 '-s': Print statistics.

'-s' prints statistics about memory usage at the end of the run.

'-sm' prints statistics about memory usage at the end of the run, just as does '-s'; it also prints information about dynamic memory allocations as they occur. '-smnnn' displays allocations of nnn bytes or more; if *nnn* is missing, 10000 is assumed.

4.2.50 '-T': Flag-setting commands for FTANGLE.

This is a family of commands that set flags appropriate only for FTANGLE.

4.2.50.1 '-TD': Permit processing of deferred macro definitions

Deferred macro definitions are '@m' (or, equivalently, '@#define') commands that appear in the code part rather than the usual definition part. These definitions are evaluated during the output (phase 2), and can cause confusion when used with the preprocessor commands, which are evaluated during the input (phase 1). Because of this confusion, deferred macro definitions are prohibited by default. To permit them, use the '-TD' option.

4.2.50.2 '-Tv': Don't print header info

By default, FTANGLE attempts to be helpful and prints some information about the command line, input and change files, etc. at the beginning of the output file. This information can be deleted by means of the '-Tv' flag. (This is done automatically when the '-F' flag is in effect, since the header information includes a time stamp that would defeat a successful file comparison.)

4.2.50.3 '-T%': Don't retain trailing comments

Unless the '-v' option is used, comments are generally deleted by FTANGLE as it writes the output file. However, in the TEX language such deletions can change the behavior of the output (by introducing extra spaces). Therefore, for TEX comments that do not begin a line are always retained unless the '-T%' command is used. This command has no effect for languages other than TEX.

4.2.51 '-t': Truncate identifiers.

The truncation option enables one to use a wider character set for identifiers than the language compiler will accept. The standard example is vanilla-flavored Fortran, which doesn't allow the underscore. If one says ''-tn6{_}'', underscores will be removed from all identifiers, then the result will be truncated to length 6. If the truncation procedure results in non-unique identifiers, these are listed.

4.2.52 '-U': Convert reserved output tokens to lower case.

Particularly during RATFOR expansion, certain output tokens such as 'DO' are produced in upper case. The '-U' option forces such tokens to be produced in lower case.

4.2.53 '-u': Undefine macro.

'-uA' undefines the WEB macro previously defined on the command line (or in '.fweb') via '-m'.

CAUTION: This command can also undefine built-in functions such as $IF. Don't do that, since built-ins can use other built-ins behind the scenes; undefining one can cause very strange behavior.

4.2.54 '-v': Make all comments verbatim.

By default, comments are not passed to the tangled output. With '-v', all comments are included verbatim in the tangled output.

4.2.55 '-W': Flag-setting commands for FWEAVE.

This is a family of commands that set flags appropriate only for FWEAVE. Commands such as '-W[' and '-Wf' can be combined as '-W[f'.

4.2.55.1 '-W[': Process bracketed array indices

4.2.55.2 '`-Wdflmvw`': Don't print various things in woven output

The printing of selected definition-part commands can be suppressed as follows:

```
-Wd --- outer definitions
-Wf --- format statements
-Wl --- limbo text definitions
-Wm --- WEB macro definitions
-Wv --- operator overloads
-Ww --- identifier overloads
```

When these commands are used, associated cross-referencing is suppressed as well.

4.2.56 '`-w`': Change name of macro package

The command '`-w`' means "Don't print '`\input fwebmac.sty`' as the first line of the '`.tex`' output file." The command '`-wfname`' means "Print '`\input fname`' as the first line."

4.2.57 '`-x`': Reduce cross-reference information.

Cross-reference information (for FWEAVE) includes the table of contents ('c'), the index ('i'), and the module list ('m'). The command '`-x`' eliminates all of that information. The command '`-x`*letters*' eliminates the piece of information corresponding to each letter in the list. For example, '`-xim`' eliminates the index and the module list.

4.2.58 '`-X`': Print selected cross-reference information.

This command is the opposite of '`-x`'. See Section 4.2.57 [-x], page 25.

4.2.59 '`-y`': Allocate dynamic memory.

This command changes the default size for a dynamically allocated memory buffer. The buffers are indicated by a one- or two-character abbreviation such as "op". The command '`-yop200`' allocates 200 units for the "op" buffer.

To query the default allocations, just say '-y'.

For a more detailed discussion of memory allocation and a menu of the various dynamic arrays, see Section 11.2.2 [Memory allocation], page 75.

4.2.60 '-Z': Display default style-file parameters.

The command '-Zabc' prints to the screen the default contents of the style-file parameters beginning with "abc". Just '-Z' prints everything.

4.2.61 '-z': Change name of style file.

The command '-znew.sty' changes the default style-file name 'fweb.sty' to 'new.sty'. The command '-z' means "Don't read any style file."

Normally the style file is read from the same directory in which the WEB file resides. To force fweb.sty to be read from the current directory, say '-z.'.

4.2.62 '-.': Don't recognize dot constants.

If this command is used, the processors will not understand that constructions such as '.LT.' are operators in Fortran or Ratfor. This command is useful if one is trying to modernize the source code to deal with symbols such as '<' instead of '.LT.'.

4.2.63 '-\': Explicitly escape continued strings.

(To be finished.)

4.2.64 '-(': Continue parenthesized strings with backslashes.

4.2.65 '-:': Set starting automatic statement number.

Useful for Fortran and Ratfor. Symbolic statement labels that are defined with the '#:0' macro command are incremented starting with the default of 90000. To change this to 789, say '-:789'.

4.2.66 '->': Redirect output.

This changes the name of the output file. If no name is given, output is redirected to the terminal.

4.2.67 '-=': Redirect output.

Equivalent to '->'.

4.2.68 '-#': Turn off comments about line and module numbers.

By default, tangled output includes comments about the line and module numbers corresponding to the current piece of code. To eliminate this clutter, say '-#'.

In some cases, bugs in tangled output, particularly Fortran, can be eliminated by using '-#'. (But please report the bug anyway; Chapter 14 [Support], page 87.)

4.2.69 '-+': Don't interpret compound assignment operators.

Both Ratfor and Fortran attempt to translate the commands '++', '+=', '--', '-=', '*=', and '/=' into code that behaves as its C counterpart. To turn this feature off, use '-+'.

4.2.70 '-/': Recognize short comments (Fortran & Ratfor).

If this command is not used, the '//' construction will be interpreted as concatenation.

One way of invoking this option is with the global language command, such as '@n/'. Another is to put the command into the initialization file '.fweb'.

4.2.71 '-!': Make '!' denote short comment (Fortran & Ratfor).

5 WEB COMMANDS

All WEB commands begin with the character '@'. It is recommended that these begin in column 1 if possible. This is required in some cases [e.g., the '@x', '@y', and '@z' in change files (see Section 3.3 [Change files], page 11), or column-oriented Fortran processing].

Some of these control codes may be used anywhere; others begin a new part. For a quick summary of the control-code mappings and to see which codes begin new parts, use the '-@' option. See Section 4.2.4 [-AT], page 13.

5.1 Debugging commands

Several commands provide localized versions of the '-1' and '-2' options related to debugging of pretty-printing.

5.1.1 '@0': Turn off debugging

This cancels the effect of a previous '@1' or '@2'. See Section 5.1.2 [AT1], page 29, Section 5.1.3 [AT2], page 29. The '@0' command should appear in a different section from the '@1' or '@2' commands.

5.1.2 '@1': Display irreducible scraps

5.1.3 '@2': Display detailed reductions of the scraps

5.2 Literal control characters

5.2.1 '@@': The character '@'.

'@@' inserts the single character '@'.

Don't forget to double the '@' even inside strings. For example,

```
puts("'@@' is represented by '@@@@'");
```

5.2.2 '@|': Literal vertical bar, or optional line break.

In TEX mode, '@|' inserts a vertical bar. This is useful inside LaTEX verbatim environments. (A simple bar would signal a shift into code mode, which is probably not what you want.)

In code mode, '@|' inserts an optional line break in an expression.

5.3 Beginning of section

5.3.1 '@ ': Begin minor section.

'@ ' begins a new minor (unstarred) section. For example,

```
@ This is an example of a minor section.  (No entry is made in the table
of contents.)
@a
main()
{}
```

5.3.2 '@*', '@*n': Begin major section.

'@*' begins a new major (starred) section (of level 0). The command must be followed by the name of the section (entry in the table of contents), followed by a period.

If '@*' is followed by a digit n, it begins a new major section of level n. For example,

```
@* MAIN PROGRAM.  This begins a major section (of level 0).
@a
main()
{}

@*1 Input routines.  Now follow some subroutines.
```

```
@a
get_input()
{}
```

5.4 Beginning of code part

The code part is begun by the appearance of either '`@a`' or '`@< Module name @>=`'.

5.4.1 '`@<`': Begin module name.

'`@<`' begins a module name, which has the form '`@< TeX text @>`'. Module names inside WEB macro definitions begin with '`@#`', not '`@<`'.

5.4.2 '`@>`': End module name.

'`@>`' ends a module name, of the form '`@< TeX text @>`'.

5.4.3 '`@A`': Begin code part of unnamed section.

'`@A`' begins the code part of an unnamed section. For example,

```
@ In an unnamed section, the code part begins with '@a'.
@a
main()
{}
```

5.4.4 '`@a`': Begin code part of unnamed section, and mark.

'`@a`' begins the code part of an unnamed section (just as does '`@A`'), and in addition marks the next unreserved identifier it finds as defined in this section. Precisely,

```
'@a' == '@A@['
```

5.5 Control codes b–z

5.5.1 '`@b`': Insert a breakpoint command.

(To be finished. Useful only for very intimate debugging of WEB codes.)

5.5.2 '`@c`': Set language to C.

See Chapter 8 [Languages], page 62, Section 8.2 [C], page 63.

5.5.3 '`@c++`': Set language to C++.

See Chapter 8 [Languages], page 62, Section 8.2.2 [Cpp], page 64.

5.5.4 '`@D`': Define outer macro.

This command begins the definition part.

'`@D`' defines an outer macro. For more discussion, Section 7.1 [Outer macros], page 48. For example,

```
@D A 1
```

5.5.5 '`@d`': Define outer macro, and mark.

This command begins the definition part.

'`@d`' defines an outer macro (just as '`@D`' does), and also marks the next identifiers as defined here. It is equivalent to

'`@d`' == '`@D@[`'

5.5.6 '@f': Format identifier or module name.

This command begins the definition part.

The construction

```
@f identifier old_identifier
```

makes FWEAVE treat *identifier* like *old_identifier*. For example,

```
@f mytype int
```

says to treat the variable `mytype` just as `int` is treated (e.g., as a reserved word in C or C++).

The *old_identifier* may be one of the following special names, which insert extra spaces according to the positions of the underscores and behave as the part of speech indicated by the base names. These are useful for dealing with macro constructions.

```
$_BINOP_
$_COMMA_
$_EXPR
$_EXPR_
$EXPR_
$UNOP_
```

When the current language is TEX, the format command can be used to change a category code according to the format

```
@f `TeXchar new_cat_code
```

5.5.7 '@i': Include file (unconditional).

If one says

```
@i test.hweb
```

the file '`test.hweb`' is inserted at the present point of the web file. By default, the current directory is searched. Files can be included from other directories by means of the `FWEB_INCLUDES` environment variable and/or the '-I' command-line option. See Section 11.1 [Environment variables], page 74, Section 4.2.17 [-I], page 16.

5.5.8 '@I': Include file (conditional).

This commands behaves like '@i' if the command-line option '-i' is not used. If it is used, then the contents of the included file is not printed in the woven output. See Section 4.2.18 [-i], page 17, Section 4.2.19 [-i!], page 17.

5.5.9 '@L': Set language.

'@Ll' sets the language to l, where l is one of {c,c++,n,n9,r,r9,x,v}. See Chapter 8 [Languages], page 62.

5.5.10 '@l': Specify limbo text.

(This command begins the definition part.)

Limbo text is material that FWEAVE should output before the start of the first section. For example,

```
@l "\\def\\A{abc}"
```

Note that '\\' stands for a backslash. In general, characters must be escaped just as in C (so that one can include things like newlines in the definitions).

Limbo text may also be typed directly into the limbo section; in that case, no escapes are necessary since one is typing ordinary TeX text. Sometime, however, the '@l' command is useful for pedagogical purposes, as the limbo material can then be defined near the point where the logical discussion is made.

5.5.11 '@M': Define WEB macro.

This command begins the definition part.

For a detailed discussion of WEB macros, see Chapter 7 [Macros], page 48.

5.5.12 '@m': Define WEB macro, and mark.

This command begins the definition part.

For a detailed discussion of WEB macros, see Chapter 7 [Macros], page 48.

5.5.13 '@N': Turn on N mode

This command must appear before the code part.

The N mode invokes language-independent behavior within the scope of a particular language. The scoping rules are the same as for language changes; i.e., using '@N' within a given section produces language-independent behavior for that section and for any modules first referenced in that section.

Fundamentally, "language-independent" behavior essentially means a literal transcription of the input to the output. For example, it inhibits blank compression by FTANGLE and tells FWEAVE to turn off "pretty-printing" (instead, the output is printed in typewriter type within a `\begin{verbatim}...\end{verbatim}` environment.

There are some subtleties with this mode (not to mention the likelihood of bugs):

1. WEB macros and built-in functions will normally be expanded even in the N mode. To inhibit expansion of a particular identifier, place '@!' before the identifier. For example,

   ```
   @
   @m A 1
   @N
   @a
   @!A = A;
   ```

   expands to `A = 1`.
2. Blank lines are significant. The N mode is ended by the appearance of the '@*' or '@ ' denoting the start of the next section. If that were preceded by one or more blank lines, those would show up in both the tangled and woven output. They might or might not be significant in the tangled output, but they almost certainly will look ugly in the woven output. To avoid this,

use the command '@%%', which deletes the remainder of the current line and all immediately following empty lines. For example,

```
@
@N
@a
x;@%%


@ Next section.
```

3. If the N mode is invoked from a compiler-like language such as Fortran, cross-referencing of variables is done as usual. However, if the language is VERBATIM (which turns on the N mode automatically), no cross-referencing is done. (Identifiers are still recognized according to FWEB's rules. Those rules as currently implemented may be essentially meaningless for some languages; in the future, provision may be made for generalizing these rules by the user.) To force an identifier to be placed into the index, precede it by '@+'.

5.5.14 '@n': Set language to Fortran–77.

See Chapter 8 [Languages], page 62.

5.5.15 '@n9': Set language to Fortran–90.

See Chapter 8 [Languages], page 62.

5.5.16 '@O': Open output file (global scope).

A statement of the form

```
@O newfile.c
```

changes the name of the FTANGLE's output file. This change remains in effect for the duration of the file, or until another '@O' is encountered. (If that occurs, the previously open file is closed.)

This command is expanded during output, so it must appear in the code part.

5.5.17 '@o': Open output file (local scope).

This behaves like '@O', except the new file name is in effect only for the current section.

5.5.18 '@r': Set language to Ratfor–77.

See Chapter 8 [Languages], page 62.

5.5.19 '@r9': Set language to Ratfor–90.

See Chapter 8 [Languages], page 62.

5.5.20 '@u': Undefine outer macro.

This command begins the definition part.

'@u' is the inverse of '@d'. For example, in C it expands to '#undef'.

5.5.21 '@v': Overload operator.

This command begins the definition part.

For a detailed discussion of overloading operators, see Section 10.2.3 [Overloading], page 71.

5.5.22 '@W': Overload identifier.

This command begins the definition part.

For a detailed discussion of overloading identifiers, see Section 10.2.3 [Overloading], page 71.

5.5.23 '@x': Terminate ignorable material, or begin material to be changed

Please see the discussion of the '@z' command. See Section 5.5.25 [ATz], page 38.

5.5.24 '@y': Begin change material

The '@y' command is permitted only in change files. See Section 3.3 [Change files], page 11.

5.5.25 '@z': Begin ignorable material, or terminate change

FWEB files may begin with the construction

```
@z
.
.
@x
```

where the '@z' and '@x' must begin in column 1. Material between the '@z' and '@x' is *pure commentary* and is ignored by both processors.

'@z' is also used in change files to end a change. See Section 3.3 [Change files], page 11.

5.6 Conversion to ASCII

Several commands are useful for generating machine-independent code. For example, FWEB works internally with the ASCII character set, so uses these commands heavily to convert from the possibly non-ASCII native character set of the machine on which FWEB is running.

5.6.1 '@'': Convert character to ASCII.

The construction '@'c'' converts 'c' to its ASCII value. In C and C++, it is converted to octal; for example, '@'A'' is output as '0101'. In Fortran and Ratfor, it is converted to decimal; the previous example would be output as '65'.

If the native character set of your machine is ASCII, the conversion will not be done unless the '-A' command-line option is used. See Section 4.2.5 [-A_], page 13.

5.6.2 '@"': Convert string to ASCII.

The construction '@"abc"' converts the enclosed string to its ASCII representation. For example, in C and C++ '@"abc"' will be output as '"\141\142\143"'.

In Fortran and Ratfor, no such simple mechanism exists in the language, so a function call is issued. For example, the previous example would be output as 'ASCIIstr('abc')'. The user is responsible for defining the function 'ASCIIstr'. The name of this function can be changed by the style-file entry ASCII_fcn. See Section 11.3.6.1 [ASCII_fcn], page 79.

If the native character set of your machine is ASCII, the conversion will not be done unless the '-A' command-line option is used. See Section 4.2.5 [-A_], page 13.

5.7 Forward referencing

5.7.1 '@[': Mark as defined.

This command marks the next (non-reserved) identifier that appears after the '@[' as being defined in the current section. It is usually issued automatically; for example, '@a' is equivalent to '@A@['.

5.8 Comments

5.8.1 '@/*': Begin long verbatim comment.

The following comment is copied to the tangled output. If you desire all comments to be so copied, use '-v'. See Section 4.2.54 [-v], page 24.

5.8.2 '@//': Begin short verbatim comment.

See the discussion of '@/*', above.

5.8.3 '@%': Ignorable comment.

If any line in a web source code contains the command '@%', all remaining material on that line (to and including the newline character) is ignored by the input driver and never processed at all.

A stronger form of this command is '@%%'. This deletes the current line as well any empty lines that immediately follow. This command is particularly useful when the N mode is in effect. See Section 5.5.13 [ATN_], page 35.

5.8.4 '@?': Begin compiler directive.

The remainder of the line is processed as a compiler directive. Optional material may be inserted automatically at the beginning of the tangled output line by means of the style-file option `cdir_start`. See Section 11.3.6 [Miscellaneous params], page 79.

5.8.5 '@(': Begin meta-comment.

Material between '@(' and '@)' is treated in the N mode. For example,

```
@(
Comment 1
Comment 2
@)
```

Style-file parameters allow optional material to be insert at the beginning and end of the meta-comment, and at the beginning of each line of output. For more information, see the style-file parameters beginning with '`meta`' (see Section 11.3.6 [Miscellaneous params], page 79).

5.8.6 '@)': End meta-comment.

See the discussion of '@(', Section 5.8.5 [ATlp], page 40.

5.9 Special brace

5.9.1 '@{': Suppress insertion of breakpoint command.

This is for detailed debugging of FWEB codes. It inserts a left brace and suppresses the insertion of a breakpoint command. See Section 5.5 [ATb], page 32.

5.10 Index entries

5.10.1 '@_': Force index entry to be underlined.

This command applies to the next identifier that appears after the '@_'. The index entry for that identifier will be underlined.

This command is usually issued automatically. For example, the index entries for the variables 'i' and 'j' in the C statement `int i,j;` will be underlined.

5.10.2 '@-': Delete index entry.

This command applies to the next identifier that appears after the '@-'; it prevents the index entry associated with that identifier from being underlined. This might be useful when the N mode is in effect.

5.10.3 '@+': Force index entry.

This command applies to the next identifier that appears after the '@+'; it forces an index entry for that identifier. It is particularly useful when the language is VERBATIM, since cross-referencing is turned off in that case.

5.10.4 '@^': Make index entry (Roman type).

To insert one's own index entry in Roman type, say '@^My entry@>'.

5.10.5 '`@.`': Make index entry (typewriter type).

To insert one's own index entry in typewriter type, say '`@^My entry@>`'.

5.10.6 '`@9`': Make index entry (\9 format).

The construction '`@9text@>`' is used to create an index entry in a format defined by the user. It is associated with the macro `\9`, which would be called during TEX's processing of the index as `\9{text}`. The user must define `\9` according to the format

```
\def\9#1{...}
```

5.11 Control text

5.11.1 '`@t`': Put control text into a TEX \hbox.

Control text is material terminated by '`@>`'; it must be all on one line and must not contain any '`@`'s. When FWEAVE sees the command '`@tcontroltext@>`', it packages the control text into an `\hbox` and ships it to the output. This command is ignored by FTANGLE.

5.11.2 '`@=`': Pass control text verbatim to the output.

For FTANGLE, the command '`@=controltext@>`' sends the control text to the output exactly as input. ('Control text' is defined in the discussion of the '`@t`' command, Section 5.11 [ATt], page 42.) FWEAVE highlights the control text by drawing a box around it.

5.12 Spacing

The spacing commands are used to refine FWEAVE's pretty-printed output. Generally it's not necessary to bother with these until you're putting the final touches on a code.

5.12.1 '@,': Insert a thin space.

Extra spacings are sometimes necessary when working with unusual macro constructions. (Need an example here.)

5.12.2 '@/': Force a line break.

For example,

```
int@/
  i,@/
  j,@/
  k;
```

5.12.3 '@|': Literal vertical bar, or optional line break.

In TeX mode, '@|' inserts a vertical bar. Here's a LaTeX example:

```
\begin{verbatim}
  The constructions @|x@| and |x| are very different.
\end{verbatim}
```

You might wish to try this out to see what FWEAVE produces.

In code mode, '@|' inserts an optional line break in an expression.

5.12.4 '@#': Blank line.

'@#' forces a line break with some extra vertical white space. Note that blank lines in the source are significant, so this command should seldom be necessary.

if '@#' is immediately followed by a letter, it is assumed that a preprocessor command is beginning. See Section 7.3 [Preprocessing], page 59.

5.12.5 '@~': Cancel line break.

For example,

```
printf("Working..."); @~ fflush(stdout);
```

5.12.6 '@&': Join items.

During FWEAVE's output, '@&' joins the items to either side with no spaces or line breaks inbetween.

This command must be distinguished from the preprocessor construction ## (paste tokens together). In a macro definition, a##bc creates the single identifier 'abc'. If one said 'a@&bc', two identifiers would be output with no spaces separating them. In simple cases, the results may look identical, but consider how things would differ if abc were itself a WEB macro.

5.13 Pseudo (invisible) operators

5.13.1 '@e': Pseudo-expression.

'@e' is an invisible expression. See Section 10.2.1 [Pseudo-operators], page 70.

5.13.2 '@;': Pseudo-semicolon.

'@;' is an invisible semicolon. These are often used in C programming to terminate a module name that expands to a compound statement. For example,

```
@c
@a
if(flag)
        @<Compound statement@>@;
else
        @<Simple statement@>;

@
```

```
@<Com...@>=
{
x;
y;
}

@
@<Sim...@>=
z
```

See Section 10.2.1 [Pseudo-operators], page 70.

5.13.3 '@:': Pseudo-colon.

'@:' is an invisible colon. See Section 10.2.1 [Pseudo-operators], page 70. It can be helpful in formatting certain C constructions correctly. For example,

```
@<Cases@>=
case 1:
case 2:
case 3@:
```

5.14 Miscellaneous commands

5.14.1 '@!': Inhibit macro expansion

FWEB macros and built-in functions are always expanded by default. This may not be desirable, particularly in the N mode. To inhibit expansion of an individual identifier, preface it by '@!'.

6 COMMENTING STYLES

FWEB allows a variety of commenting styles. The visible comments are in the font `\cmntfont`, which defaults to `\tenrm`.

6.1 Invisible comments

```
@z...@x --- If a source or include file begins with '@z', then all
            material is skipped until and including a line beginning in
            column 1 with '@x'.

@%       --- All material until and including the next newline is
             completely ignored.

@%%      --- As '@%', but also skip blank lines that immediately
              follow the current line.
```

For example,

```
@z
Author:  J. A. Krommes
@x
@c @% This sets the global language to C.
@* EXAMPLE.
```

6.2 Visible comments

'`/* ... */`' is a long comment (may extend over several lines).

'`// ...`' is a short comment (terminated by next newline).

'`@(...@)`' is a meta-comment. Meta-comments are a localized form of the N mode. Tangled meta-comments are begun by the contents of the style-file entry '`meta.top`' and terminated by '`meta.bottom`'. Each line of the meta-comment is begun by '`meta.prefix`'. Woven meta-comments are begun by '`meta_code.begin`' and ended by '`meta_code.end`'. See Section 11.3.6 [Miscellaneous params], page 79.

```
@n
```

```
@a
        program main
/* Get input. */
        call get_input // Read the parameter file.
/* Process information.  Comments like this
   can be split over several lines. */
@(
Meta-comments provide a poor-man's alignment feature
 i --- counter
 x --- data value
@)
        i = 1
        x = 2.0
        call exec(i,x)
        end
```

The use of meta-comments is not recommended. Use ordinary long comments instead. Inside of them, use the various powerful features of TeX or LaTeX (such as `\halign`) to format your comment appropriately.

7 MACROS and PREPROCESSING

FWEB recognizes two kinds of macros: *outer macros*, and *WEB macros* or *inner macros*. Control codes associated with either of these kinds normally begin the definition part. However, WEB macros are sometimes allowed in the code part as well; Section 7.2 [WEB macros], page 49.

7.1 Outer macros

Outer macros are defined by '`@d`'. These may be placed in any definition part. They are collected during phase 1; during phase 2, they are simply copied in order of their appearance to the beginning of the output file. This is most useful for C or C++ codes; it's a shorthand way of typing '`#define`' when the positioning of the '`#define`' is unimportant.

As an example,

```
@c
@
@d YES 1
@d NO 0
@a
main()
{}

@
@d BUF_LEN 100
@a
...
```

The keyword into which the '`@d`' is translated is language-dependent; it is controlled by the style-file parameter '`outer_def`'. See Section 11.3.6 [Miscellaneous params], page 79.

Outer macros can be undefined by '`@u`'. The translation is controlled by the style-file parameter '`outer_undef`'. See Section 11.3.6 [Miscellaneous params], page 79.

7.2 WEB macros

WEB macros are defined by '`@m`'. These should normally be placed in the definition section. They are collected during FTANGLE's phase 1 and placed at the top of the unnamed section, so they are all known during the output in phase 2.

In unusual situations when macros are being conditionally defined and/or undefined, the order of processing a macro definition becomes significant. If the command-line option '`-TD`' is used, then WEB macros may be used in the code part as well; they are then called deferred macros. These definitions will be processed during phase 2 in the order that the code sections are processed, which may not be the same as the physical order in the source file.

WEB macros are often used in conjunction with the WEB preprocessor commands. Preprocessor commands are always processed during phase 1, so they do not interact properly with deferred macros. It is for this reason that deferred macros are normally prohibited from appearing in the code part.

7.2.1 Various features of WEB macros

- WEB macros with a variable number of arguments are indicated by an ellipsis, as in

  ```
  @m VAR(x,y,z,...) text
  ```

- Adjacent strings in macro text are automatically concatenated.

7.2.2 Special tokens

The following special tokens may be used in the text of WEB macro definitions:

7.2.2.1 ANSI C-compatible tokens

```
##          --- Paste tokens on either side to form a new identifier.
#parameter  --- Convert parameter to string (without expansion).
```

7.2.2.2 Extensions to ANSI C macro syntax

(In the following list, the forms '`#{n}`' and '`#[n]`' may not work correctly in complicated situations. This is a design deficiency that will be corrected someday.)

```
#*parameter --- Like #parameter, but pass a quoted string through unchanged.
#!parameter --- Don't expand argument.
#'parameter --- Convert parameter to a single-quoted string (no expansion).
#"parameter --- Convert parameter to a double-quoted string (no expansion).
#0          --- Number of variable arguments.
#n          --- n-th variable argument, counting from 1.
#{0}        --- Like #0, but the argument may be a macro expression
                known at run time.
#{n}        --- Like #n, but the argument may be a macro expression.
#[0]        --- The total number of arguments (fixed + variable).  (The
                argument inside the brackets may be a macro expression.)
#[n]        --- The n-th argument (including the fixed ones), counting
                from 1.  (The argument inside teh brackets may be a
                macro expressions.)
#.          --- Comma-separated list of all variable arguments.
#:0         --- Unique statement number (expanded in phase 1).
#:nnn       --- Unique statement number for each invocation of this
                macro (expanded in phase 2).
#<          --- Begin a module name.
#,          --- Internal comma; doesn't delimit macro argument.
```

7.2.3 Built-in functions

Built-in functions are essentially macros, but they sometimes implement functions that a user could not define. They all begin with a dollar sign and are in upper case.

In the original FWEB design, built-in functions began with an underscore. This usage conflicts with the conventions for reserved words in ANSI C. In fact, FWEB understands built-in function names that begin with either a dollar sign or an underscore. If you don't like the underscore convention for some name, you can undefine it, as in

```
@#undef _A
```

No user-defined macro should begin with an underscore or a dollar sign! It might interfere with the functioning of some internal built-in function.

7.2.3.1 `$A`: Convert to ASCII.

`$A`(*string*) is the built-in equivalent of `@'...'` or `@"..."`. (See see Section 5.6 [ATquote], page 38 and see Section 5.6.2 [AT"], page 39.) Note the extra parentheses required by the built-in.

7.2.3.2 `$ABS`: Absolute value.

`$ABS`(*expression*) returns the absolute value of the macro expression.

7.2.3.3 `$ASSERT`: Assert a condition.

`$ASSERT`(*expression*) evaluates the macro expression. If the expression is false, an error message is printed and the run aborts.

This built-in is useful for ensuring that WEB macros required by the code are properly initialized. It must appear in the *code part* (not the definition part).

7.2.3.4 `$COMMENT`: Generate a comment.

`$COMMENT`(*string*) generates a comment in the output file.

This built-in is sometimes useful in conjunction with the processing of WEB macros, since ordinary comments are removed when macros are processed.

7.2.3.5 `$DATE`: Today's date.

`$DATE` generates a string consisting of the date in the form `"August 16, 2001"`.

7.2.3.6 `$DAY`: The day.

`$DAY` generates a string consisting of the day of the week, such as `"Monday"`.

7.2.3.7 `$DECR`: Decrement a macro.

`$DECR`(N) redefines the numeric macro N to be one less than its previous value. (If N does not simplify to a number, an error results.) In other words, in the language of C the effect is to say `N--`.

The two-argument form `$DECR`(N,m) executes the equivalent of `N -= m`.

7.2.3.8 `$DEFINE`: Deferred macro definition.

`$DEFINE` (to be finished).

7.2.3.9 `$DO`: Macro do loop.

`$DO`(*macro,imin,imax[,di]*)`{...}` repetitively defines *macro* as would the FORTRAN statement `do macro = imin,imax,di`. For example,

```
$DO(I,0,2)
  {
  a[I] = I;
  }
```

generates the three statements

```
  a[0] = 0;
  a[1] = 1;
  a[2] = 2;
```

Do not assume that the macro remains defined after the end of the iteration.

Instead of the delimiting braces, parentheses may be used. These may be useful to help FWEAVE format certain constructions correctly.

Nested delimiters are handled correctly. The delimiters are required even only a single statement is to expanded.

7.2.3.10 `$DUMPDEF`: Dump macro definitions to the terminal.

In the call `$DUMPDEF`(*m1,m2,...*), *m1*, *m2*, and so on are macro calls (with arguments if appropriate). Two lines of output are generated for each argument. Line 1 is the macro definition; line 2 is its expansion using the provided arguments.

You can use this built-in to debug your own macros, or to find out the secrets of FWEB's built-ins.

7.2.3.11 `$E`: Base of natural logarithms.

The expression `$E` returns e to the default machine precision. The expression `$E(iprec)` returns *e* to the decimal precision *iprec* (which must be less than 50).

7.2.3.12 `$ERROR`: Send error message to output.

`$ERROR`(*string*) prints an error message in FWEB's standard form.

7.2.3.13 `$EVAL`: Evaluate a macro expression.

`$EVAL`(*expression*) used FWEB's expression evaluator to reduce the macro expression to its simplest form. An attempt to perform arithmetic on combinations of non-macro identifiers and numbers generates a warning message.

7.2.3.14 `$EXP`: Exponential function.

`$EXP(x)` returns *e* to the power *x*.

7.2.3.15 `$GETENV`: Get value of environment variable.

`$GETENV`(*name*) returns a string consisting of the current value of the environment variable *name*. (Under VMS, logical names behave like environment variables.)

7.2.3.16 `$HOME`: The user's home directory.

`$HOME` is equivalent to `$GETENV(HOME)`.

7.2.3.17 `$IF`: Two-way conditional.

(To be finished.)

7.2.3.18 `$IFCASE`: n-way conditional.

7.2.3.19 `$IFDEF`: Two-way conditional.

7.2.3.20 `$IFNDEF`: Two-way conditional.

7.2.3.21 `$IFELSE`: Two-way conditional.

7.2.3.22 `$INCR`: Increment a macro.

`$INCR`(N) redefines the numeric macro N to be one greater than its previous value. (If N does not simplify to a number, an error results.) In other words, in the language of C the effect is to say `N++`.

The two-argument form `$INCR`(N,m) executes the equivalent of `N += m`.

7.2.3.23 `$INPUT_LINE`: Line number that begins current section.

`$INPUT_LINE` is the number of the line in the WEB source file that begins the current section.

7.2.3.24 `$L`: Change to lower case.

`$L`(*string*) changes *string* to lower case.

7.2.3.25 `$LANGUAGE`: Identifier for current language.

This expands to an identifier that denotes the current language, as follows:

```
Language    $LANGUAGE
 C             $C
 C++           $CPP
 Fortran       $N
 Fortran-90    $N90
 Ratfor        $R
 Ratfor-90     $R90
 TeX           $X
 VERBATIM      $V
```

Note that these are identifiers, not WEB macros. They are intended to be used in '`$IF`' or '`$IFELSE`' statements such as

```
$IF($LANGUAGE==$C, C-text, other-text)
```

For multiway switches, the '`$LANGUAGE_NUM`' built-in is more useful; Section 7.2.3.26 [$LANGUAGE_NUM], page 55.

7.2.3.26 `$LANGUAGE_NUM`: Number of current language.

'`$LANGUAGE_NUM`' expands to an integer that uniquely defines the current language, as follows:

```
Language  $LANGUAGE_NUM
 C             0
 C++           1
 Fortran       2
 Fortran-90    3
 Ratfor        4
 Ratfor-90     5
 TeX           6
 VERBATIM      7
```

This built-in is useful in conjunction with an '`$IFCASE`' construction; Section 7.2.3.18 [$IFCASE], page 54.

7.2.3.27 `$LEN`: Length of string.

`$LEN`(*string*) returns the length of *string* in bytes. If *string* is not surrounded by quotes, it is interpreted as if it were quoted (so it is not expanded if it is a macro). Thus, in the example

```
@m SS string
$LEN(SS)
```

the value returned is 2, not 5.

7.2.3.28 `$LOG`: Natural logarithm.

`$LOG(x)` returns the natural logarithm of x.

7.2.3.29 `$LOG10`: Logarithm to the base 10.

`$LOG10(x)` returns the logarithm to the base 10 of x.

7.2.3.30 `$M`: Define a macro.

`$M` is equivalent to `$DEFINE`. See Section 7.2.3.8 [$DEFINE], page 52.

7.2.3.31 `$MAX`: Maximum of a list.

`$MAX(x1,x2,...)` returns the minimum of the list of arguments. (There must be at least one argument.)

(To be finished.)

7.2.3.32 `$MIN`: Minimum.

`$MIN(x1,x2,...)` returns the minimum of the list of arguments. (There must be at least one argument.)

7.2.3.33 `$MODULE_NAME`: Name of present WEB module.

7.2.3.34 `$MODULES`: Total number of independent modules.

`$MODULES` gives the total number of independent modules—that is, the number of independent module names, plus 1 for the unnamed module.

7.2.3.35 `$OUTPUT_LINE`: Current line number of tangled output.

7.2.3.36 `$P`: The C preprocessor symbol '#'.

`$P` is a synonym for `$UNQUOTE("#")`. See Section 7.2.3.49 [$UNQUOTE], page 59. It is useful for constructing WEB macro definitions that expand to C preprocessor statements. For example,

```
@m CHECK(flag)
        $P if(flag)
                special code;
        $P endif
```

7.2.3.37 `$PI`: Pi.

The expression `$PI` returns *pi* to the default machine precision. The expression `$PI(iprec)` returns *pi* to the decimal precision *iprec* (which must be less than 50).

7.2.3.38 `$POW`: Exponentiation.

`$POW`(x,y) generates x raised to the power y.

7.2.3.39 `$ROUTINE`: Current function (Ratfor only).

When Ratfor is the current language, `$ROUTINE` expands to a string built of the name of the current program, function, or subroutine. This function is not useful for other languages, for which it expands to the null string.

7.2.3.40 `$SECTION_NUM`: Number of current web section.

`$SECTION_NUM` returns an integer greater than 0 that is the number of the current web section. (This is not the LaTeX section number such as 3.4.)

7.2.3.41 `$SECTIONS`: Maximum section number.

`$SECTIONS` is the maximum section number as understood by fweave.

7.2.3.42 `$SQRT`: Square root.

`$SQRT(x)` return the square root of x.

7.2.3.43 `$STRING`: Expand, then stringize.

`$STRING`(s) expands its argument if it is a macro, then makes the expansion into a quoted string. If the argument is already a quoted string, it is returned unchanged.

7.2.3.44 `$STUB`:

(To be finished.)

7.2.3.45 `$TIME`: The time.

`$TIME` returns a string consisting of the local time in the form `"19:59"`.

7.2.3.46 `$TRANSLIT`: Transliteration.

(To be finished.)

7.2.3.47 `$U`: Change to upper case.

`$U`(*string*) changes *string* to upper case.

7.2.3.48 `$UNDEF`: Undefine a macro.

`$UNDEF`(*macro*) undefines a WEB macro.

7.2.3.49 `$UNQUOTE`: Remove quotes from string.

`$UNQUOTE`(*string*) returns *string* without its surrounded quotes.

7.2.3.50 `$VERBATIM`: (Obsolete)

This was an old name for `$UNQUOTE`. See Section 7.2.3.49 [$UNQUOTE], page 59.

7.2.3.51 `$VERSION`: Present FWEB version number.

`$VERSION` returns a string built out of the FWEB version number, such as `"1.32"`.

7.3 Preprocessing

FWEB preprocessor commands may appear in either the definition or the code parts. But BEWARE: No matter where they appear, they are expanded during INPUT, not output. (This is probably a design flaw.) For more discussion, Section 7.2 [WEB macros], page 49.

```
@#define identifier --- Define a WEB macro; equivalent to `@m'.
@#undef identifier  --- Undefine a WEB macro.

@#ifdef identifier  --- Is WEB macro defined?  Equivalent to
                        `@#if defined identifier'.
@#ifndef identifier --- Is WEB macro not defined?  Equivalent to
                        `@#if !defined identifier'.

@#if expression
@#elif expression
@#else
@#endif
```

In the '**@#if**' statement, the *expression* may contain WEB macros, but must ultimately evaluate to a number. If that number is zero, the expression is false; otherwise, it is true.

The *expression* following constructions such as '**@#if**' is evaluated by a built-in expression evaluator that can also be used for other purposes, such as in macro expansion. Its behavior is again motivated by expression evaluation in ANSI C; it is not quite as general, but should be more than adequate. (One design flaw that will be fixed someday is that the order of expression evaluation is not necessarily left-to-right, as it is in C.) It supports both integer and floating-point arithmetic (with type promotion from integer to floating-point if necessary), and the ANSI `defined` operator. Operators with the highest precedence (see table below) are evaluated first; as usual, parentheses override the natural order of evaluation. The unary operator `defined` has the highest precedence; all the other unary operators have the next highest (and equal) precedence; then come the binary operators. When the operator exists in C, the action taken by FWEB is precisely that that the C compiler would take. Arithmetic is done in either **long** or **double** variables, as implemented by the C compiler that compiled FTANGLE. (This was the easy choice, not necessarily the most desirable one.)

The operators, listed from highest precedence to lowest, are as follows (printed documentation only):

Unary operators:

`defined` — `defined` is a unary operator that acts on identifier tokens. '`defined id`' or equivalently '`defined(id)`' evaluates to 1 (true) if the identifier is defined as a WEB macro; to 0 (false) otherwise. The construction '`@#if defined A`' works the same way as '`@#ifdef A`', but you can use '`defined`' in expressions, as in

'`@#if defined(A) || defined(B)`'.

(The parentheses around the macro names are optional.) With the advent of '`defined`', the WEB preprocessor statements '`@#ifdef`' and '`@#ifndef`' become redundant, but are often useful shorthands.

`-` — Unary minus.

`!` — Logical `NOT`. `!`*expr* evaluates to 0 if *expr* is nonzero, and evaluates to 1 if *expr* is 0.

`~` — One's complement of an integer. For example, `~0` $= -1$.

Binary operators:

`^^` — Exponentiation (all languages). 2`^^`$3 = 8$.

`^`, `**` — Exponentiation (FORTRAN or RATFOR).

`*`, `/`, `%` — Multiplication, division, and modulus: *a* `%` *b* means *a* mod *b*; for example, `5 % 3 = 2`.

`+`, `-` — The usual plus and minus.

`<<` — *a* `<<` *b* means shift integer *a* left *b* bits. $1 \ll 3 = 8$.

`>>` — As above, but right-shift. $7 \gg 2 = 1$.

`<`, `<=`, `>`, `>=` — Evaluates to 1 if the inequality holds, to 0 otherwise. E.g., `(2.0 < 3.0)` evaluates to 1.

`==`, `!=` — *a*`==`*b* (*a*`!=`*b*) evaluates to 1 (0) if *a* equals *b*; evaluates to 0 (1) otherwise.

`&` — Bitwise `AND`. The truth table is `0b1100 & 0b1010 = 0b1000`.

`^` — Bitwise `EXCLUSIVE OR` (C). (For FORTRAN, use '`.xor.`'.) The truth table is `0b1100 .xor. 0b1010 = 0b0110`.

`|` — Bitwise `OR`. The truth table is `0b1100 | 0b1010 = 0b1110`.

`&&` — Logical `AND`. *a* `&&` *b* evaluates to 1 if both *a* and *b* are true (nonzero).

`||` — Logical `OR`. *a* `||` *b* evaluates to 1 if either *a* or *b* are true.

Note in particular the use of the single caret, which is language-dependent: it is an exponentiation operator for FORTRAN (just as in TeX), but the exclusive or operator for C. Also, note that the bitwise operators should almost never be used. For logic, almost always you will be using '`==`', '`!=`', '`&&`', and '`||`'.

8 LANGUAGES

FWEB has the ability to work with more than one language during a single run. The language in effect at the beginning of the first section defines the *global language*. Further language changes within a section have scope local to that section.

Usually, 'language' means a compiler language like Fortran or C. These languages will be "pretty-printed" by FWEAVE. Pretty-printing can be inhibited by turning on the N mode (globally, with the command-line option '`-N`'; locally, with '`@N`') or by selecting the VERBATIM 'language'; in both of these cases, the input text is echoed literally to the output of both FTANGLE and FWEAVE.

'Language' is a stronger concept than 'mode'. For example, when a language is selected, the extension of the tangled output file is changed appropriately—for example, if '`test.web`' contains C code (that is, contains the command '`@c`'), '`test.web`' tangles into '`test.c`' (compressing blanks and otherwise making the tangled output relatively unreadable) and FWEAVE pretty-prints using the C syntax. Turning on the N mode does not affect the language; FTANGLE copies the source code literally into '`test.c`' (no blank compression or other modifications), and FWEAVE typesets the source code within a verbatim environment (no pretty-printing). When the VERBATIM language is selected, the N mode is turned on automatically, but FTANGLE writes its output to a file with a special default extension that can be customized in the style file. See Section 11.3.6 [Miscellaneous params], page 79.

8.1 Setting the language

The most general form of a language command is

`@`[L]*ltext*[*options*]

where *l* is a language symbol, *text* is converted into a hyphenated option, and *options* have the same syntax as on the command line. The language symbols must be in lower case; they are

```
C          --- c
C++        --- c++
Fortran-77 --- n
Fortran-90 --- n9
Ratfor-77  --- r
Ratfor-90  --- r9
TeX        --- x
```

```
VERBATIM   --- v
```

Thus, for example,

```
@n9[-n&]
```

means set the language to Fortran–90 and use free-form syntax with the ampersand as the continuation character.

A language command should appear somewhere in limbo, before the start of the first section. The language in effect at the beginning of the first section defines the global language.

Language commands may be used within sections, but the new language remains in force only for that section. The language of a named module is inherited from the language in effect at the time the name is first used. Thus, in the following example, the global language is Fortran–77, but an arbitrary number of C functions can be placed into a C-language module with just one '`@c`' language-changing command.

```
@n
@
@a
        program main
        end

@c
@<C@>@;

@
@<C@>=
int fcn()
{}
```

FTANGLE will write two output files for this example.

8.2 Special hints and considerations for each language

8.2.1 C

Special remarks. (To be completed.)

8.2.2 C++

Special remarks. (To be completed.)

8.2.3 Fortran

Special remarks. (To be completed.)

8.2.4 Ratfor

Special remarks. (To be completed.)

8.2.5 TeX

Special remarks. (To be completed.)

8.2.6 Verbatim

Special remarks.

9 RATFOR

“Ratfor” stands for “RATional FORtran”. More...

9.1 Ratfor commands

9.1.1 Ratfor–77 commands

```
break;
case i:
default:
do ...; {...}
else {...}
for(a;b;c) {...}
if(condition) {...}
next;
repeat {...} until(condition);
return expression;
switch(expression) {...}
while(condition) {...}
```

9.1.2 Additional Ratfor–90 commands

```
contains:
interface name {...}
interface operator(op) {...}
interface assignment(assgnmnt) {...}
module name {...}
private:
sequence:
type name {...}
where(expression) {...}
```

9.2 Caviats about Ratfor

The version of Ratfor built into FWEB differs slightly from its Unix counterpart:

1. Numeric statement labels must be followed by a colon; they should be first on their line.
2. The quoting convention for characters and strings follows that of C: Single-quote single characters, double-quote strings.
3. In a **switch**, cases fall through to the next case unless terminated by **break** (just as in C).
4. The **do** statement must be terminated by a semicolon if followed by a simple statement. (It's unnecessary if followed by a left brace that begins a compound statement.)
5. Use `&&` and `||` for the logical AND and OR.
6. Do not use an **end** statement at the very end of a program unit; it is added automatically when the closing brace is sensed.

10 FORMATTING

FWEB uses TEX to format its documentation. Either Plain TEX or LaTEX may be used. FWEB's "new look" beginning with Version 1.40 is very flexible, but works only with LaTEX.

10.1 Typesetting

FWEAVE works in conjunction with either TEX or LaTEX. It is always possible to create an adequately typeset document by processing with TEX. However, FWEB's "new look" (beginning with version 1.40) is designed to work with LaTEX; it probably will not be retrofitted to TEX unless days become longer than 24 hours. The new look is more book-like, following ideas from Briggs' `nuweb`.

To inform FWEAVE you will be using TEX as the formatter, just say 'FWEAVE `test`' or 'FWEAVE `-PT test`'. To inform it you will be using LaTEX, say 'FWEAVE `-PL test`'.

If you will always be using a particular processor, you should probably put '`+PL`' into your '`.FWEB`' file (see Section 11.2 [Initialization], page 75).

10.1.1 FWEAVE's OUTPUT

When one says 'FWEAVE `test`', the file '`test.tex`' is created. The TEX commands contained in this file are organized into several sequential groups.

1. `\input` command to read in FWEAVE's macro package.

   By default, the initial input command is '`\input fwebmac.sty`'. The name of the macro package can be changed with the '`-w`' command-line option. See Section 4.2.56 [-w], page 25.
2. `\Wbegin` command (sets up the environment for TEX or LaTEX).

   The `\Wbegin` macro sets up certain defaults. These can be overridden in the limbo section.
3. Limbo text from the style-file parameter `limbo`. See Section 11.3.6.9 [limbo], page 80.
4. Beginning of user's limbo section.
5. TEX commands for individual WEB sections.
6. `\input` command to read in the index data file.
7. `\input` command to read in the module-list data file.

8. `\Winfo` command (summarizes some status information).
9. `\Wcon` command (generates the table of contents, and ends the run).

10.1.2 LaTeX support

Original LaTeX support (through version 1.30) was substantially incomplete in that LaTeX's `\output` routine was usurped by the relatively simple one used for FWEB's TeX support. However, beginning with version 1.40, full LaTeX support is provided. LaTeX's `\output` routine is used, as are its sectioning commands (with minor changes), table-of-contents facilities, etc.

10.1.2.1 LaTeX's document style

FWEAVE uses `\documentstyle{article}`. The document style can be changed by the style-file option '`LaTeX.style`'. See Section 11.3.5 [Fwebmac params], page 78. However, FWEAVE has not been tested with other document styles.

To incorporate style options—i.e., to obtain the effect of '`\documentstyle[myoptions]{article}`', use the style-file option `LaTeX.options`, as in

```
LaTeX.options "myoptions"
```

TeX commands in the user's limbo section will appear after the `\documentstyle` and `\begin{document}` commands, so may be used to redefine macro definitions in the style file.

10.1.2.2 Sections in LaTeX

FWEB's sectioning commands ('`@*`', '`@*`'*n*, and '`@ `') are converted into LaTeX's section commands such as `\section` (n=0), `\subsection` (n=1), and `\subsubsection` (n=2). During LaTeX's processing of the tex file, it keeps track of the maximum depth achieved by '`@*n`'. This number is written as the last item in the '`.aux`' file. During the next LaTeX run, that number is used to map the '`@ `' commands to the next most insignificant sectioning command. That level of sectioning command is slightly redefined from LaTeX's default, so don't try to redefine it.

The '`.aux`' file is also used by both processors to generate appropriate error messages that refer to the LaTeX section number instead of the internal one.

10.1.2.3 LaTeX's table of contents

In essence, the table of contents is produced by the LaTeX commands

```
\pagenumbering{roman}
\topofcontents
\maketitle
\tableofcontents
\botofcontents
\newpage
```

By default, the FWEB hooks `\topofcontents` and `\botofcontents` are empty, but they may be used in special circumstances to override the usual behavior. One can set the parameters for \maketitle in the limbo section in the usual LaTeX way, except that one is encouraged to use FWEB's \Title macro instead of `\title`:

```
\Title{MYCODE.WEB}
\author{My name}
\date{January 1, 2001}
```

By default, FWEAVE uses

```
\title{}%
\author{}%
\date{\today\\[3pt]\Time}%
```

10.1.2.4 Customizing LaTeX's output

Several flags are provided to change the appearance of the final LaTeX document. These are

- \ifpagerefs (index references by pages or section numbers).
- \numberTeX (number the beginning of the TeX part).
- \numberdefs (number the beginning of the definition part).
- \numbercode (number the beginning of the code part).

The defaults for these flags are

```
\pagerefsfalse
\numberTeXfalse
```

```
\numberdefstrue
\numbercodetrue
```

10.1.3 Page references

When one says '`\pagerefstrue`' (LaTeX only), index references are made by page numbers rather than module numbers or LaTeX section numbers. If there is more than one section per page, they are identified by '`a`', '`b`', '`c`', etc., such as '`section 17b`'.

The information necessary to process page references in this way is written into the '`.aux`' file. As usual with LaTeX, several runs may be required for the references to be fully consistent with the source file.

(To be finished.)

10.2 Pretty-printing

Pretty-printing refers to FWEAVE's attempt to typeset and highlight the code in a readable way. This is usually done automatically for all of the compiler-like languages such as C. However, it can be inhibited by turning on the N mode with '`@N`' or by using the VERBATIM language (selected with '`@Lv`').

(More...)

10.2.1 Pseudo-operators

Pseudo-operators behave like a particular part of speech for the purposes of FWEAVE's formatting, but are invisible on output. The pseudo-operators are

```
@e --- pseudo-expression.  See Section 5.13 [ATe], page 44.
@; --- pseudo-semicolon.  See Section 5.13.2 [AT;], page 44.
@: --- pseudo-colon.  See Section 5.13.3 [ATcolon], page 45.
```

10.2.2 Alternatives for various input tokens

FWEAVE translates various input constructions into allegedly more readable symbols—for example, in FORTRAN it translates '`.LT.`' into '`<`'.

Here is a table of what you can type on input, and what FWEAVE will typeset. The first entry is standard Fortran; the parenthesized material is an allowable input alternative. (In most cases, the pretty input alternatives follow C's convention.)

`.lt.` (`<`)	$\to <$	`.or.` (`||`)	$\to \lor$
`.le.` (`<=`)	$\to \leq$	`.neqv.`	$\to \not\equiv$
`.eq.` (`==`)	$\to \equiv$	`.xor.`	$\to \not\equiv$
`.ne.` (`!=,<>`)	$\to \neq$	`.eqv.`	$\to$?=
`.gt.` (`>`)	$\to >$	`.not.` (`!`)	$\to \neg$
`.ge.` (`>=`)	$\to \geq$	`**` (`^`)	$\to$ (a+b)^(c+d) $\to (a+b)^{c+d}$
`.and.` (`&&`)	$\to \land$	`//` (`\/`)	$\to \|$

These same conventions are allowed in RATFOR mode. Note that in FORTRAN and RATFOR '`//`' is interpreted by default as the concatenation symbol, not the start of a short comment. To override that default, use one of the command-line options '`-n/`', '`-r/`', or '`-/`', or use a language-changing command of the form '`@n/`'.

10.2.3 Overloading operators and identifiers

For special effects in the woven output, there are commands to help one change the appearance of operators and identifiers.

10.2.3.1 Overloading operators

A feature common to both C++ and FORTRAN–90 is *operator overloading*, the ability to extend or redefine the definition of an operator such as '`.FALSE.`' or '`=`'. FORTRAN–90 even allows one to define new 'dot' operators—for example, one might define the operator '`.IN.`' to test for inclusion in a set. In a nontrivial extension of the original design, FWEAVE allows one to define how overloaded operators should appear on output. For example, it is much more readable to read "**if**$(x \in set)$" than "**if**(x `.IN.` *set*)". Indeed, this feature can be used even when the language does not permit overloading in order to customize the appearance of the woven output.

The '`@v`' control code is used to change the appearance of an operator. The format is

```
@v new_operator_symbol_or_name "TeX material" old_operator
```

This means "Display the new operator according to the *TEX material*, but treat it like the old operator—e.g., unary or binary—for formatting purposes. The quoted TEX material is treated just like a C string, so for example if you want to include a backslash you must escape it with another backslash. For example, we can make an equals sign display on output as a large left arrow by saying

```
@v = "\\Leftarrow" =
```

Two \Fortran\ examples are

```
@v .FALSE. "\.{.FALSE.}" .FALSE.
@v .IN. "\\in" +
```

This feature can go a long way toward enhancing readability of the woven output, particularly when operators are actually being overloaded. It can also lead to arbitrarily bizarre output that no-one else will understand. As usual, restraint is advised.

10.2.3.2 Overloading identifiers

Although operator overloading is quite useful, it does not allow one to change the appearance of identifiers. In its most general form, such a facility becomes quite complicated; one must endow FWEAVE with a macro-processing facility analogous to that of FTANGLE. This has not been done yet (probably it will be someday). In the meantime, one has the command '`@W`', which provides a restricted form of such a facility. *This command, new with version~1.30, is experimental, and not firmly established. Changes in usage and/or syntax may be made in future versions.*

The most general form of the '`@W`' command is

```
@W identifier "replacement text"
```

This means: Replace any references to *identifier* in the woven output with the *replacement text*.

A more restrictive form is

```
@W identifier \newmacro
```

which replaces references to *identifier* with a call to `\newmacro`. (Note that there are no quotes in this form.)

The shortest form is

```
@W identifier .
```

which replaces references to *identifier* with a call to `\identifier`. For example, the identifier *x* normally appears in woven output as '`\.{\Wshort\{x\}}`'. If one says

```
@W x .
```

one will instead get the macro reference '`\x`', which could be defined to give a variety of special effects.

One of the important uses of this facility is to expedite special formatting of array references. This subject is discussed separately below in the section on "Special array formatting," where an example is given.

10.2.4 Information options

Several of the command-line options can be used to elicit information about the initial state of FWEB.

'`-@`' displays information about the control codes. See Section 4.2.4 [-AT], page 13.

'`-D`' displays information about reserved words. See Section 4.2.10 [-D_], page 14.

'`-y`' displays default dynamic memory allocations. See Section 4.2.59 [-y], page 25.

'`-Z`' displays default values of style-file parameters. See Section 4.2.60 [-Z_], page 26.

11 CUSTOMIZATION

The default behavior of FWEB can be changed in a variety of ways.

1. Unix environment variables (logical variables in VMS) affect path or file names.
2. An initialization file resides in the home directory.
3. A style file resides in the current directory.

The initialization file (usually called '`.fweb`') is intended to contain command-line options (one per line) that are to be used in every run.

The style file is intended to provide more local customization, perhaps differing for each run. The style file does not contain command-line options; rather, it contains parameter settings that override FWEB's defaults. The '`-p`' option (see Section 4.2.38 [-p], page 21) provides a means of incorporating style-file entries into '`.fweb`'.

The order of processing is:

1. Evaluate environment variables.
2. Read '`.fweb`'.
3. Process '`.fweb`' options beginning with '`+`'.
4. Read and process command-line options (except for '`-p`').
5. Process remaining '`.fweb`' options.
6. Process any '`-p`' options.
7. Read and process the style file.

11.1 Environment variables

`FWEB_INCLUDES` — Colon-delimited list of directories to search for include files. (One can append to this list by means of the '`-I`' option; see Section 4.2.17 [-I], page 16.)

`FWEB_INI` — Name of the initialization file. If not defined, either '`.fweb`' or '`fweb.ini`' is chosen, depending on the machine. The initialization file always resides in `$HOME`.

`FWEB_STYLE_DIR` — Directory in which style file resides. If not defined, the current directory is used.

11.2 Initialization

11.2.1 The initialization file

On startup, FWEB attempts to read an initialization file. This always resides in the user's HOME directory. It is usually called '`.fweb`' ('`fweb.ini`' on personal computers). The default file name can be overridden by the environment variable `FWEB_INI`.

One may put into '`.fweb`' any option that might be used as a command-line option. (Presently, there must be just one entry per line.) If the option begins with a '`+`', it is processed before the actual command-line options; otherwise, it is processed after. Generally, '`.fweb`' options should begin with '`+`' so that one may override them from the command line.

11.2.2 Memory allocation

The command-line option '`-y`' (see Section 4.2.59 [-y], page 25) is used to change the default allocation for a dynamic array. The arrays have a one- or two-character abbreviation denoted by *aa*. Some error messages will use this abbreviation when suggesting that you increase a default allocation. To query the present allocations of variable aa, just say '`-y`'*aa*. To query everything, say '`-y`'.

(The detailed explanations for these arrays aren't here yet. The on-line info has a terse list.)

11.3 The Style file

A style file (default name '`fweb.sty`') may reside in the user's current directory (or directory specified by the environment variable `FWEB_STYLE_DIR`). The default name can be changed by the command-line option '`-z`' (see Section 4.2.61 [-z], page 26).

The style file is processed after all command-line options have been processed. Note that the command-line option '`-p`' (see Section 4.2.38 [-p], page 21) buffers up style-file entries. Effectively, these are treated as residing in a temporary file that is read just before the local style file. Thus, '`-p`' options placed in '`.fweb`' behave as 'global' style-file entries and will be overridden by a matching entry in the local style file.

Style-file entries have the form

```
keyword [=] value
```

For example,

```
LaTeX.options = "eqalign"
```

The syntax is completely free-form.

The descriptions of the parameters aren't completed yet. Please see the reference guide.

11.3.1 Customizing FWEAVE's index

11.3.1.1 `index`

`index.tex` is the name of the file into which the index is written. The character '#' is translated into the root name of the web file.

`index.preamble` are TeX commands that begin the index.

`index.postamble` are TeX commands that end the index.

`index.collate` specifies the collating sequence for the index.

11.3.1.2 `delim`

`delim_0` is the string to insert after the identifier in an index entry.

`delim_n` is the string to insert between two section numbers in an index entry.

11.3.1.3 `group_skip`

`group_skip` is a string of TeX commands to insert between letter groups.

11.3.1.4 item_0

item_0 is the TeX command to begin an index entry.

11.3.1.5 language

language.prefix begins a language entry.; language.suffix ends one.

11.3.1.6 lethead

lethead.prefix begins a letter group; lethead.suffix ends one. The flag lethead.flag controls the format of the letter group: if it is zero, nothing is inserted; if it is positive, an upper-case letter is inserted; if it is negative, a lower-case letter is inserted.

11.3.1.7 underline

underline.prefix is the TeX command to begin an underlined index entry.

underline.suffix is the TeX command to end an underlined index entry.

11.3.2 Customizing the module list

modules.tex is the name of the file into which the module names are written.

modules.preamble is a string of TeX commands to begin the list of modules.

modules.postamble is a string of TeX commands to end the list of modules.

modules.info is the name of the TeX macro that formats the command line and related information.

11.3.3 Customizing the table of contents

`contents.tex` is the name of the file into which the table of contents is written.

`contents.preamble` is the TeX string that begins printing the table of contents.

`contents.postamble` is the TeX string that ends the table of contents.

11.3.4 Customizing cross-reference subscripts

When FWEAVE pretty-prints code, it can attach cross-reference subscripts to various kinds of identifiers. The following flags select which identifiers are so subscripted.

11.3.5 Customizing the behavior of '`fwebmac.sty`' macros

To some extent, the behavior of FWEB's macro package '`fwebmac.sty`' can be changed by means of the following parameters, instead of actually editing '`fwebmac.web`' itself.

11.3.5.1 `format`

The `format` fields specify the macro to use to pretty-print various kinds of identifiers, as follows: (not finished).

11.3.5.2 `indent`

`indent.TeX` specifies paragraph indentation for the TeX part.

`indent.code` specifies similar indentation for the code part.

When running under LaTeX, the documents is (effectively) begun by the command `\documentstyle[options]{style}`. The options field can be specified by `LaTeX.options`; the style field by `LaTeX.style`.

(To be finished.)

11.3.6 Miscellaneous style-file parameters

11.3.6.1 ASCII_Fcn

See Section 5.6.2 [AT"], page 39.

11.3.6.2 cchar

Continuation character for Fortran code output.

11.3.6.3 cdir_start

This parameter has the form `cdir_start.l`, where l is one of `C`, `Cpp`, `N`, `N90`, `R`, `R90`, `X`, or `V`. The contents of this parameter is written immediately after the '`@?`' that begins a compiler directive.

11.3.6.4 meta (FTANGLE)

`meta.top.l` specifies text that precedes material enclosed by '`@(...@)`'.

`meta.prefix.l` begins each line of the meta-comment.

`meta.bottom.l` specifies text that follows the meta-comment.

11.3.6.5 outer

FTANGLE converts '`@d`' to `outer.def`, and '`@u`' to `outer.undef`.

11.3.6.6 protect

The strings `protect.l` specify the protection character(s) to end a continued line.

11.3.6.7 suffix

The extension for the files output by FTANGLE is specified by `suffix.l`.

11.3.6.8 macros

The default name of the macro package to be read in. (This can be overridden by the command-line option '`-w`'; see Section 4.2.56 [-w], page 25.)

11.3.6.9 limbo

TEX material to be printed at the beginning of the limbo part, just before the text from '`@l`' commands. See Section 5.5.10 [ATl], page 34.

11.3.6.10 meta (FWEAVE)

11.3.6.11 preamble

Additional TEX material can be inserted at the beginning of a named section with `preamble.named` and at the beginning of an unnamed one with `preamble.unnamed`.

11.3.6.12 dot_constant

In Fortran, 'dot' constants such as `.LT.` are begun and ended by periods. In special circumstances, the beginning and ending characters may be modified by `dot_constant.begin.l` and `dot_constant.end.l`.

11.3.6.13 null_file

The name of the null file. For more discussion, see Section 3.3 [Change files], page 11.

11.3.7 Automatic file name completion:

For more information, see Section 4.2.13 [-e], page 15.

12 USAGE TIPS and SUGGESTIONS

In this section we collect various tips and suggestions to help one make full use of FWEB. *(There's more to come here!)*

12.1 Converting an existing code to FWEB

In summary, to convert an existing code to FWEB, you should do the following. (The following simple procedure assumes that you put all the subroutines into the unnamed module. However, other more elaborate schemes are possible.)

1. Place invisible commentary about the author, version, etc. at the beginning of the source file by bracketing it with '@z...@x'. The '@z' must be the first two characters of the file.
2. Next, set the language by including a command such as '@n'.
3. Place an '@a' command before each program unit (e.g., main **program**, **subroutine**, or **function**).
4. Before each '@a', place an '@*' or '@ ' command, followed by TeX documentation about that particular section of code.
5. If you have program units longer than about twelve lines, either make them function calls, if you can afford the overhead and can impart sufficient information via the function name, or break them up into shorter fragments by using named modules. Insert the command '@<Name of module@>' in place of the fragment you're replacing, then put that fragment somewhere else, prefaced by '@ ' and '@<Name of module@>='.
6. Make sure your comments are valid TeX. (You can't have things like raw underscores or dollar signs in comments, since those cause TeX to take special actions.)
7. Beautify and clarify your documentation by using code mode (enclosing stuff between vertical bars) liberally within your TeX.
8. After you've seen the woven output, you may need to go back and format a few identifiers or section names so that FWEAVE understands them properly, or you may need to insert some pseudo-semicolons ('@;'), pseudo-expressions ('@i'), or pseudo-colons ('@:').
9. Consider using the built-in macro preprocessor to make your code more readable—for example, replace raw numerical constants by symbolic names.
10. If you are a FORTRAN user, for ultimate readability consider converting to RATFOR. The initial annoyance is getting rid of column~6 continuations. With the aid of a good editor, this can be done simply. For example, in 'emacs' one can replace the regular expression [carriage return, five spaces, something not equal to space, tab, or 0] with [backslash, carriage return, six spaces]:

```
M-x replace-regexp RET
C-q C-j \.{\ \ \ \ \ }[\^\.\ tab 0]RET
\\\\ C-q C-j \.{\ \ \ \ \ \ }RET
```

Get rid of the keywords such as **then** or **end if** in favor of braces. Change singly-quoted character strings to doubly-quoted ones.

12.2 Programming tips and other suggestions

This section will be enlarged in the future! Meanwhile, please feel free to contact '`krommes@princeton.edu`' for help and advice, and to suggest items to include here.

1. Periodically check `ftp.pppl.gov:/pub/fweb/READ_ME` for bug reports and other news. Make bug reports! See Chapter 14 [Support], page 87.
2. Most options in '`.fweb`' should begin with '`+`' so they can be overridden by command-line options for the job itself. See Section 11.2 [Initialization], page 75
3. Put standard command-line options into '`.fweb`'. Also put there standard style parameters—e.g.,

   ```
   +pindex.tex "#.ndx"
   +pmodules.tex "#.mds"
   +pcontents.tex "#.cts"
   ```

4. Learn how to use the style file. See Section 11.3 [Style], page 75.
5. Use the info options '`-@`', '`-D`', '`-y`', and '`-Z`' to find out about various internal FWEB tables (control codes, reserved words, memory allocations, and style-file parameters). See Section 10.2.4 [Info options], page 73.
6. Begin all FWEB sources with invisible commentary bracketed by '`@z...@x`'. See Section 5.5.25 [ATz], page 38.
7. Always include an explicit language-setting command in the limbo section. See Chapter 8 [Languages], page 62.
8. Keep sections quite short. Knuth suggests a dozen lines. That's quite hard to achieve sometimes, but almost never should a section be more than a page long.
9. It's easy to define macros from the command line to expedite conditional preprocessing. See Section 4.2.22 [-m], page 18.
10. Use the preprocessor construction '`@#if 0...@#endif`' to comment out unwanted code. See Section 7.3 [Preprocessing], page 59.
11. For logical operations with the preprocessor, use '`||`', not '`|`'.
12. It's conventional to identify the ends of long preprocessor constructions as follows:

```
@#if A
.
.
@#endif // |A|
```

13. To debug an errant WEB macro, use the built-in function '`$DUMPDEF`'. See Section 7.2.3.10 [$DUMPDEF], page 53.
14. Use '`@?`' for compiler directives. See Section 5.8.4 [AT?], page 40. Use the style-file parameters '`cdir_start`' to specify information that will be written out at the beginning of the line.
15. Stick to the standard FWEB commenting style '`/*...*/`' or '`//...`'. Don't use alternatives such as FORTRAN's column 1 convention; these may not work or may not be supported someday. See Chapter 6 [Comments], page 46.
16. The meta-comment feature '`@(...@)`' provides a poor-man's alignment feature. But that's not in the spirit of TeX; learn to use '`\halign`' or the LaTeX alternatives.
17. In FORTRAN, use '`#:0`' to declare readable alphabetic statement labels. See Chapter 7 [Macros], page 48
18. When mixing languages, define the language of a module at the highest possible level—e.g., in the unamed module, not after '`@<...@>=`'.

13 NEW FEATURES, version 1.40

1. *The meaning of '`@+`' has changed.* (SORRY!) Formerly, this inhibited a line break; that function is now performed by '`@~`'. The new meaning of '`@+`' is to force an index entry (the opposite of '`@-`', which inhibits an index entry).

   If you have large codes using the old '`@+`' that you do not wish to convert, you can recover the old mappings by placing the following commands into '`fweb.sty`':

   ```
   yes_index = "~"
   no_line_break = "+"
   ```

   However, please try to make the conversion; the new codes are intended to be more symmetrical and easier to remember.

2. *Built-in functions now begin with '`$`', not '`_`'.* The underscore prefix was a bad design decision; it introduces conflicts with ANSI C in certain circumstances. To ease conversion, the old forms are still understood. Thus, one can use '`$EVAL`' and '`_EVAL`' interchangably. However, the underscore forms may be deleted in future releases.

3. *Full LaTeX support.* FWEB no longer usurps LaTeX's `\output` routine, and LaTeX's sectioning commands, table-of-contents commands, etc. are used. The appearance of the woven output is changed to be more book-like. (This is an experiment.)

4. *Verbatim language.* '`@Lv`' selects a language-independent format. See Section 8.2.6 [Verbatim], page 64

5. *Language-independent mode.* The N mode inhibits pretty-printing, blank compression, etc.; source code is essentially copied literally from input to output. This mode is turned on automatically by the VERBATIM language, but it can also be used with the other languages. It is turned on by the command-line option '`-N`' or the local command '`@N`'. See Section 5.5.13 [ATN_], page 35.

6. *Writing of temporary files.* When the '`-F`' command-line option is in effect, tangled output is written to temporary files instead of the final target files, and the temporary files are compared to the last version of the target files on disk. If there is no change, the target files are not updated. This avoid unnecessary recompilation if only the documentation, not the code, was changed. See Section 4.2.14 [-F_], page 15.

7. *Converting output tokens to lower case.* See Section 4.2.52 [-U_], page 24.

8. *The built-in functions '`$E`' and '`$PI`'.* See Section 7.2.3.11 [$E], page 53, Section 7.2.3.37 [$PI], page 57.

9. *The built-in functions '`$EXP`', '`$LOG`', and '`$LOG10`'.* See Section 7.2.3.14 [$EXP], page 53, Section 7.2.3.28 [$LOG], page 56, and Section 7.2.3.29 [$LOG10], page 56.

10. *'`$MAX`' and '`$MIN`' generalized to take arbitrary list of arguments.* See Section 7.2.3.31 [$MAX], page 56, Section 7.2.3.32 [$MIN], page 57.

11. *The marriage-saver option.* In response to a serious user request, see See Section 4.2.6 [-B_], page 14.

14 SUPPORT

FWEB is supported by John Krommes, '`krommes@princeton.edu`'. This project is a *spare-time activity*! I try for, but cannot promise, reasonably quick (one week) response to questions. Bug reports submitted with very short test files will be verified. For very simple fixes, a change file will be provided. Generally, however, bugs are not fixed until the next release. Releases occur intermittently, depending on my other professional obligations.

Suggestions are very welcome. Many of FWEB's current features were incorporated in response to users' requests. However, the queue for future improvements is long; nothing may happen immediately.

Marcus Speh moderates an FWEB FAQ. See '`/pub/fweb/faq`'.

Index

The index is obviously not complete; the following are tests.

(Index is nonexistent)

Short Contents

FWEB Copying Permissions 1
1 INTRODUCTION to FWEB 2
2 WEB CONCEPTS 4
3 FILES 9
4 RUNNING FWEB 12
5 WEB COMMANDS 29
6 COMMENTING STYLES 46
7 MACROS and PREPROCESSING 48
8 LANGUAGES 62
9 RATFOR 65
10 FORMATTING 67
11 CUSTOMIZATION 74
12 USAGE TIPS and SUGGESTIONS 82
13 NEW FEATURES, version 1.40 85
14 SUPPORT 87
Index 88

Table of Contents

FWEB **Copying Permissions** 1

1 INTRODUCTION to FWEB 2
- 1.1 History of WEB and literate programming 2
- 1.2 FWEB features 2

2 WEB CONCEPTS 4
- 2.1 The FWEB processors 4
- 2.2 The structure of a web 4
- 2.3 Modules 6
  - 2.3.1 The unnamed module 6
  - 2.3.2 Named modules 6
- 2.4 Phases of processing 7
  - 2.4.1 The phases of FTANGLE 7
  - 2.4.2 The phases of FWEAVE 8

3 FILES 9
- 3.1 Input files 9
  - 3.1.1 Automatic file-name completion 9
- 3.2 Output files 10
- 3.3 Change files 11

4 RUNNING FWEB 12
- 4.1 Command-line syntax 12
- 4.2 Command-line options 12
  - 4.2.1 Negating options 13
  - 4.2.2 ‘`-1`’: Turn on brief debugging mode (FWEAVE). 13
  - 4.2.3 ‘`-2`’: Turn on verbose debugging mode (FWEAVE). 13
  - 4.2.4 ‘`-@`’: Display the control-code mappings. 13
  - 4.2.5 ‘`-A_`’: Turn on ASCII translations. 13
  - 4.2.6 ‘`-B`’: Turn off audible beeps. 14
  - 4.2.7 ‘`-b`’: Number blocks. 14
  - 4.2.8 ‘`-c`’: Set global language to C. 14
  - 4.2.9 ‘`-c++`’: Set global language to C++. 14
  - 4.2.10 ‘`-D`’: Display reserved words. 14
  - 4.2.11 ‘`-d`’: Convert do...enddo. 15
  - 4.2.12 ‘`-E`’: Change the delimiter of a file-name extension. 15

4.2.13 '-e': Turn on automatic file-name completion. 15
4.2.14 '-F': Compare output files with old versions. 15
4.2.15 '-f': Turn off module references for identifiers. 16
4.2.16 '-h': Get help. 16
4.2.17 '-I': Append to search list for include files. 16
4.2.18 '-i': Don't print '@I' include files. 17
4.2.19 '-i!': Don't read '@I' include files. 17
4.2.20 '-L': Select global language. 17
4.2.21 '-l': Echo input line. 17
4.2.22 '-m': Define WEB macro. 18
4.2.23 '-m4': Understand m4 built-in commands. 18
4.2.24 '-m;': Append pseudo-semicolons. 18
4.2.25 '-n': Set global language to Fortran–77. 18
4.2.26 '-n9': Set global language to Fortran–90. 18
4.2.27 '-n;': Supply automatic semicolons (Fortran). 19
4.2.28 '-nb': Number ifs and dos (Fortran). 19
4.2.29 '-np': Print semicolons (Fortran). 19
4.2.30 '-n\': Free-form syntax continued by backslash. 19
4.2.31 '-n&': Free-form syntax continued by ampersand. 20
4.2.32 '-n/': Recognize short comments (Fortran). 20
4.2.33 '-n!': Make '!' denote short comment (Fortran). 20
4.2.34 '-n)': Reverse array indices (Fortran) 20
4.2.35 '-o': Don't overload operators. 20
4.2.36 '-q': Don't translate Ratfor. 21
4.2.37 '-P': Select TEX processor. 21
4.2.38 '-p': Buffer up a style-file entry. 21
4.2.39 '-r': Set global language to Ratfor–77. 21
4.2.40 '-r9': Set global language to Ratfor–90. 21
4.2.41 '-rg': Set goto parameters. 21
4.2.42 '-k': Suppress comments about Ratfor translation. 22
4.2.43 '-K': Write comments about Ratfor translation. 22
4.2.44 '-r;': Turn on auto-semi mode (Ratfor). 22
4.2.45 '-rb': Number ifs and dos (Ratfor). 22
4.2.46 '-r/': Recognize short comments (Ratfor). 22
4.2.47 '-r!': Make '!' denote short comment (Ratfor). 22
4.2.48 '-r)': Reverse array indices (Ratfor) 22
4.2.49 '-s': Print statistics. 23
4.2.50 '-T': Flag-setting commands for FTANGLE. 23
4.2.50.1 '-TD': Permit processing of deferred macro definitions 23
4.2.50.2 '-Tv': Don't print header info 23
4.2.50.3 '-T%': Don't retain trailing comments 23
4.2.51 '-t': Truncate identifiers. 24

4.2.52 '-U': Convert reserved output tokens to lower case...... 24
4.2.53 '-u': Undefine macro..................................... 24
4.2.54 '-v': Make all comments verbatim..................... 24
4.2.55 '-W': Flag-setting commands for FWEAVE............. 24
4.2.55.1 '-W[': Process bracketed array indices 24
4.2.55.2 '-Wdflmvw': Don't print various things in woven output 25
4.2.56 '-w': Change name of macro package 25
4.2.57 '-x': Reduce cross-reference information............... 25
4.2.58 '-X': Print selected cross-reference information......... 25
4.2.59 '-y': Allocate dynamic memory....................... 25
4.2.60 '-Z': Display default style-file parameters.............. 26
4.2.61 '-z': Change name of style file......................... 26
4.2.62 '-.': Don't recognize dot constants.................... 26
4.2.63 '-\': Explicitly escape continued strings............... 26
4.2.64 '-(': Continue parenthesized strings with backslashes... 26
4.2.65 '-:': Set starting automatic statement number......... 27
4.2.66 '->': Redirect output................................ 27
4.2.67 '-=': Redirect output................................ 27
4.2.68 '-#': Turn off comments about line and module numbers. 27
4.2.69 '-+': Don't interpret compound assignment operators... 27
4.2.70 '-/': Recognize short comments (Fortran & Ratfor)..... 27
4.2.71 '-!': Make '!' denote short comment (Fortran & Ratfor). 28

5 WEB COMMANDS 29

5.1 Debugging commands 29
5.1.1 '@0': Turn off debugging 29
5.1.2 '@1': Display irreducible scraps 29
5.1.3 '@2': Display detailed reductions of the scraps........... 29
5.2 Literal control characters 29
5.2.1 '@@': The character '@'................................ 29
5.2.2 '@|': Literal vertical bar, or optional line break.......... 30
5.3 Beginning of section 30
5.3.1 '@ ': Begin minor section.............................. 30
5.3.2 '@*', '@*n': Begin major section........................ 30
5.4 Beginning of code part 31
5.4.1 '@<': Begin module name............................. 31
5.4.2 '@>': End module name............................... 31
5.4.3 '@A': Begin code part of unnamed section............... 31
5.4.4 '@a': Begin code part of unnamed section, and mark..... 31

5.5 Control codes b–z 32
- 5.5.1 '@b': Insert a breakpoint command. 32
- 5.5.2 '@c': Set language to C. 32
- 5.5.3 '@c++': Set language to C++. 32
- 5.5.4 '@D': Define outer macro. 32
- 5.5.5 '@d': Define outer macro, and mark. 32
- 5.5.6 '@f': Format identifier or module name. 33
- 5.5.7 '@i': Include file (unconditional). 33
- 5.5.8 '@I': Include file (conditional). 34
- 5.5.9 '@L': Set language. 34
- 5.5.10 '@l': Specify limbo text. 34
- 5.5.11 '@M': Define WEB macro. 34
- 5.5.12 '@m': Define WEB macro, and mark. 35
- 5.5.13 '@N': Turn on N mode 35
- 5.5.14 '@n': Set language to Fortran–77. 36
- 5.5.15 '@n9': Set language to Fortran–90. 36
- 5.5.16 '@O': Open output file (global scope). 36
- 5.5.17 '@o': Open output file (local scope). 37
- 5.5.18 '@r': Set language to Ratfor–77. 37
- 5.5.19 '@r9': Set language to Ratfor–90. 37
- 5.5.20 '@u': Undefine outer macro. 37
- 5.5.21 '@v': Overload operator. 37
- 5.5.22 '@W': Overload identifier. 37
- 5.5.23 '@x': Terminate ignorable material, or begin material to be changed 38
- 5.5.24 '@y': Begin change material 38
- 5.5.25 '@z': Begin ignorable material, or terminate change 38

5.6 Conversion to ASCII 38
- 5.6.1 '@'': Convert character to ASCII. 38
- 5.6.2 '@"': Convert string to ASCII. 39

5.7 Forward referencing 39
- 5.7.1 '@[': Mark as defined. 39

5.8 Comments 39
- 5.8.1 '@/*': Begin long verbatim comment. 39
- 5.8.2 '@//': Begin short verbatim comment. 40
- 5.8.3 '@%': Ignorable comment. 40
- 5.8.4 '@?': Begin compiler directive. 40
- 5.8.5 '@(': Begin meta-comment. 40
- 5.8.6 '@)': End meta-comment. 40

5.9 Special brace 41
- 5.9.1 '@{': Suppress insertion of breakpoint command. 41

5.10 Index entries 41
- 5.10.1 '@_': Force index entry to be underlined. 41

5.10.2 '@-': Delete index entry. 41
5.10.3 '@+': Force index entry. 41
5.10.4 '@^': Make index entry (Roman type). 41
5.10.5 '@.': Make index entry (typewriter type). 42
5.10.6 '@9': Make index entry (\9 format). 42
5.11 Control text 42
5.11.1 '@t': Put control text into a TeX \hbox. 42
5.11.2 '@=': Pass control text verbatim to the output. 42
5.12 Spacing 42
5.12.1 '@,': Insert a thin space. 43
5.12.2 '@/': Force a line break. 43
5.12.3 '@|': Literal vertical bar, or optional line break. 43
5.12.4 '@#': Blank line. 43
5.12.5 '@~': Cancel line break. 44
5.12.6 '@&': Join items. 44
5.13 Pseudo (invisible) operators 44
5.13.1 '@e': Pseudo-expression. 44
5.13.2 '@;': Pseudo-semicolon. 44
5.13.3 '@:': Pseudo-colon. 45
5.14 Miscellaneous commands 45
5.14.1 '@!': Inhibit macro expansion 45

6 COMMENTING STYLES 46

6.1 Invisible comments 46
6.2 Visible comments 46

7 MACROS and PREPROCESSING 48

7.1 Outer macros 48
7.2 WEB macros 49
7.2.1 Various features of WEB macros 49
7.2.2 Special tokens 49
7.2.2.1 ANSI C-compatible tokens 49
7.2.2.2 Extensions to ANSI C macro syntax 50
7.2.3 Built-in functions 50
7.2.3.1 $A: Convert to ASCII. 51
7.2.3.2 $ABS: Absolute value. 51
7.2.3.3 $ASSERT: Assert a condition. 51
7.2.3.4 $COMMENT: Generate a comment. 51
7.2.3.5 $DATE: Today's date. 51
7.2.3.6 $DAY: The day. 51
7.2.3.7 $DECR: Decrement a macro. 52
7.2.3.8 $DEFINE: Deferred macro definition. 52

7.2.3.9 $DO: Macro do loop. 52
7.2.3.10 $DUMPDEF: Dump macro definitions to the terminal. 53
7.2.3.11 $E: Base of natural logarithms. 53
7.2.3.12 $ERROR: Send error message to output. 53
7.2.3.13 $EVAL: Evaluate a macro expression. 53
7.2.3.14 $EXP: Exponential function. 53
7.2.3.15 $GETENV: Get value of environment variable. . . 53
7.2.3.16 $HOME: The user's home directory. 54
7.2.3.17 $IF: Two-way conditional. 54
7.2.3.18 $IFCASE: n-way conditional. 54
7.2.3.19 $IFDEF: Two-way conditional. 54
7.2.3.20 $IFNDEF: Two-way conditional. 54
7.2.3.21 $IFELSE: Two-way conditional. 54
7.2.3.22 $INCR: Increment a macro. 54
7.2.3.23 $INPUT_LINE: Line number that begins current section. 54
7.2.3.24 $L: Change to lower case. 55
7.2.3.25 $LANGUAGE: Identifier for current language. 55
7.2.3.26 $LANGUAGE_NUM: Number of current language. 55
7.2.3.27 $LEN: Length of string. 56
7.2.3.28 $LOG: Natural logarithm. 56
7.2.3.29 $LOG10: Logarithm to the base 10. 56
7.2.3.30 $M: Define a macro. 56
7.2.3.31 $MAX: Maximum of a list. 56
7.2.3.32 $MIN: Minimum. 57
7.2.3.33 $MODULE_NAME: Name of present WEB module. 57
7.2.3.34 $MODULES: Total number of independent modules. 57
7.2.3.35 $OUTPUT_LINE: Current line number of tangled output. 57
7.2.3.36 $P: The C preprocessor symbol '#'. 57
7.2.3.37 $PI: Pi. 57
7.2.3.38 $POW: Exponentiation. 57
7.2.3.39 $ROUTINE: Current function (RATFOR only). . . 58
7.2.3.40 $SECTION_NUM: Number of current WEB section. 58
7.2.3.41 $SECTIONS: Maximum section number. 58
7.2.3.42 $SQRT: Square root. 58
7.2.3.43 $STRING: Expand, then stringize. 58
7.2.3.44 $STUB: 58

7.2.3.45 $TIME: The time. 58
7.2.3.46 $TRANSLIT: Transliteration. 59
7.2.3.47 $U: Change to upper case. 59
7.2.3.48 $UNDEF: Undefine a macro. 59
7.2.3.49 $UNQUOTE: Remove quotes from string. 59
7.2.3.50 $VERBATIM: (Obsolete) 59
7.2.3.51 $VERSION: Present FWEB version number. 59
7.3 Preprocessing 59

8 LANGUAGES 62

8.1 Setting the language 62
8.2 Special hints and considerations for each language 63
8.2.1 C 63
8.2.2 C++ 64
8.2.3 Fortran 64
8.2.4 Ratfor 64
8.2.5 TeX 64
8.2.6 Verbatim 64

9 RATFOR 65

9.1 Ratfor commands 65
9.1.1 Ratfor–77 commands 65
9.1.2 Additional Ratfor–90 commands 65
9.2 Caviats about Ratfor 65

10 FORMATTING 67

10.1 Typesetting 67
10.1.1 FWEAVE's OUTPUT 67
10.1.2 LaTEX support 68
10.1.2.1 LaTEX's document style 68
10.1.2.2 Sections in LaTEX 68
10.1.2.3 LaTEX's table of contents 69
10.1.2.4 Customizing LaTEX's output 69
10.1.3 Page references 70
10.2 Pretty-printing 70
10.2.1 Pseudo-operators 70
10.2.2 Alternatives for various input tokens 71
10.2.3 Overloading operators and identifiers 71
10.2.3.1 Overloading operators 71
10.2.3.2 Overloading identifiers 72
10.2.4 Information options 73

11 CUSTOMIZATION . . . 74

11.1 Environment variables . . . 74
11.2 Initialization . . . 75
11.2.1 The initialization file . . . 75
11.2.2 Memory allocation . . . 75
11.3 The Style file . . . 75
11.3.1 Customizing FWEAVE's index . . . 76
11.3.1.1 `index` . . . 76
11.3.1.2 `delim` . . . 76
11.3.1.3 `group_skip` . . . 76
11.3.1.4 `item_0` . . . 77
11.3.1.5 `language` . . . 77
11.3.1.6 `lethead` . . . 77
11.3.1.7 `underline` . . . 77
11.3.2 Customizing the module list . . . 77
11.3.3 Customizing the table of contents . . . 78
11.3.4 Customizing cross-reference subscripts . . . 78
11.3.5 Customizing the behavior of '`fwebmac.sty`' macros . . . 78
11.3.5.1 `format` . . . 78
11.3.5.2 `indent` . . . 78
11.3.6 Miscellaneous style-file parameters . . . 79
11.3.6.1 `ASCII_Fcn` . . . 79
11.3.6.2 `cchar` . . . 79
11.3.6.3 `cdir_start` . . . 79
11.3.6.4 `meta` (FTANGLE) . . . 79
11.3.6.5 `outer` . . . 79
11.3.6.6 `protect` . . . 79
11.3.6.7 `suffix` . . . 80
11.3.6.8 `macros` . . . 80
11.3.6.9 `limbo` . . . 80
11.3.6.10 `meta` (FWEAVE) . . . 80
11.3.6.11 `preamble` . . . 80
11.3.6.12 `dot_constant` . . . 80
11.3.6.13 `null_file` . . . 80
11.3.7 Automatic file name completion: . . . 81

12 USAGE TIPS and SUGGESTIONS . . . 82

12.1 Converting an existing code to FWEB . . . 82
12.2 Programming tips and other suggestions . . . 83

13 NEW FEATURES, version 1.40 . . . 85

14 SUPPORT **87**

Index **88**