FWEB

A WEB system of structured documentation
for multiple languages

By John A. Krommes

FWEB

This Texinfo documentation describes FWEB Version 1.61.

- To learn about new features of this version, see Section 14.1 [V1.61], page 124.
- For a quick introduction to, and review of the structure of an FWEB source file, see Section 2.2 [Structure], page 5.
- If you used to receive e-mail information about FWEB but don't any longer, it's probably because you need to update your e-mail address in the `fweb-users` mailing list. Subscription instructions can be found in Chapter 15 [Support], page 131.
- Bug reports and suggestions are much appreciated, but are no longer acknowledged individually. See Chapter 15 [Support], page 131.
- The next major release, FWEB Version 2.00, is planned for no earlier than January 1, 2000.

This documentation is now accessible on the World-Wide Web from

`http://w3.pppl.gov/~krommes/fweb_toc.html`.

Other sources of information about FWEB are the archival files of the `fweb-users` and `fweb-installers` mailing lists. To learn how to obtain those, see Chapter 15 [Support], page 131.

If you are learning FWEB for the first time, you will probably find that this (unfinished) manual is not sufficiently pedagogical. For background, please refer to Knuth's book cited in Chapter 1 [Intro], page 3. You should also browse through Chapter 2 [Concepts], page 5, in particular Section 2.2 [Structure], page 5.

FWEB Copying Permissions

FWEB is “free.” This means that everyone is free to use them and free to redistribute them on a free basis. FWEB operates under the terms of the GNU General Public License; see, for example, section “Distribution” in *The GNU Emacs Manual*.

Although it is hoped that FWEB will be useful, there is *ABSOLUTELY NO WARRANTY*.

1 INTRODUCTION to FWEB

FWEB is a system for *literate programming*. It enables one to maintain both documentation and source code in a single place (the `web` file), and to explain the code in terms of a *web* of very small fragments. Because FWEB is intimately integrated with TEX, one gains many advantages such as book-quality typesetting and extensive cross-referencing facilities. A simple example program is described in Section 2.2 [Structure], page 5.

FWEB was originally intended for scientific programming (the 'F' stands for FORTRAN), 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 (both F77 and F90), RATFOR, and (in a limited fashion) TEX itself. It also attempts to implement a WYSIWYG language-independent mode as well as a (closely-related but not identical) verbatim 'language'. *The language-independent features are highly experimental* and are not recommended.

The origins and philosophy 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 Stanford Junior University, 1992).

Knuth's original WEB was written in Pascal, and it formatted Pascal code. Silvio Levy introduced CWEB, a WEB system written in C for C. FWEB is a (by now, substantial) modification of version 0.5 of CWEB that was graciously supplied by Levy. It also borrows various ideas from the works of Ramsey and Briggs on language-independent webs.

The original WEB's worked with Plain TEX. More recently, many users have turned to Lamport's LaTEX because of its ease of use and higher-level features. Excellent and extensive development of LaTEX has been accomplished, as described by Goossens, Mittelbach, and Samarin, *The LaTEX Companion* (Addison–Wesley, Reading, MA, 1994). The present version of FWEB is intended to be used with LaTEX (LaTEX2e, in particular); Plain TEX is no longer supported.

1.1 History of WEB and literate programming

(To be completed; see Knuth's book, cited in Chapter 1 [Intro], page 3.)

1.2 Features of FWEB

FWEB is distinguished from its relatives in several respects:

- FWEB introduces the concept of a *current language* (see Chapter 8 [Languages], page 83), so more than one compiler language can be processed in a single FWEB run. For example, mixtures of C++ and FORTRAN are common in modern scientific programming.
- FWEB understands the syntaxes of several of the more important compiler languages: C, C++, FORTRAN (both F77 and F90), 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 89.
- FWEB has a built-in C-like *macro preprocessor*. This is especially useful for FORTRAN and RATFOR, which have no predefined preprocessor. However, certain extensions such as variable numbers of arguments make the FWEB preprocessor sometimes useful even for C and C++. See Chapter 7 [Macros], page 62 and Section 7.3 [Preprocessing], page 80.
- Many aspects of FWEB's behavior, default strings, etc. can be customized by means of setting parameters in a `makeindex`-like *style file* (by default, '`fweb.sty`'). See Section 12.3 [Style], page 112.

2 WEB CONCEPTS

The principle concepts of WEB programming are laid out in Knuth's book, the reference to which was given in Chapter 1 [Intro], page 3. 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: FWEAVE and FTANGLE

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 (in principle) be processed with either TeX or LaTeX. (If a file '`test.tex`' already exists, FWEAVE will ask for confirmation before overwriting it if it does not think that the file was created by a previous run of FWEAVE.)

The output file produced by FTANGLE is not intended for human eyes (or for editors!); it is for compiling only. All changes to the code should be made to the `web` file, since changes made directly to the output file would be overwritten the next time the `web` source is tangled. In an attempt to discourage messing with FTANGLE's output file, all unnecessary spaces are deliberately removed.

A common way of integrating FWEB into ones program development is to do all compilations through a `make` file, into which one puts an extra dependency line that explains how to produce the compilable output file from the `web` source. For example,

```
test.c: test.web
        ftangle test

test.o: test.c
        gcc -c test test.c
```

With this approach, one is not so tempted to edit '`test.c`'.

FWEB development is now based on LaTeX; Plain TeX is no longer supported. For detailed descriptions of the LaTeX support, see Section 10.1.3 [LaTeX], page 93.

2.2 The structure of a web

An FWEB source file is structured into *sections*, which correspond to logical subunits of the code (either a function or a fragment of a function). Each section consists of three *parts*, each of which is optional: the

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

When FTANGLE outputs code, it can combine the code parts of (possibly noncontiguous) sections into larger units called *modules*, as explained in Section 2.3 [Modules], page 8.

With the aid of sections, one's possibly huge and logically complex code can be broken down into bite-sized pieces, each one easily comprehensible. Since sections may correspond to only a small part of a function or subroutine, 1000-line main programs (they still exist!) should become a thing of the past.

Since sections can be combined into modules, there is no need for sections that must be physically contiguous in the output file to be contiguous in the source file. This allows for great flexibility in structuring the documentation of the code.

2.2.0.1 A simple example

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

```
@n/ % Set FWEB language to Fortran, and recognize short // comments.

\Title{example.web} % \Title is an FWEB TeX macro.
\author{J. A. Krommes} % \author is a LaTeX macro.

@* INTRODUCTION.
This code is intended to illustrate the use of the |write| statement.
It also provides a simple example of the \FWEB\ macro preprocessor.

@m A_CONSTANT 1.2345 // \FWEB\ preprocessor macro definition.

@a
        program main
        call compute
        end

@ The computational routine is pretty boring.
@a
        subroutine compute
        write(*,*) 'Macro value = ', A_CONSTANT
        end

@* \INDEX.
```

Commands to FWEB are begun by the '`@`' symbol (see Chapter 5 [AT commands], page 38). In this example, the first command, '`@n`', sets the global language to FORTRAN-77. One should always begin one's code with a language-setting command.

In this example, the language command is invoked with an optional argument '`/`'. That is necessary in FORTRAN in order to tell FWEB to use the short (single-line) comment form beginning with '`//`', which otherwise conflicts with the concatenation operator. See Section 4.2.40 [-n/], page 27.

For more information about languages, see Chapter 8 [Languages], page 83. For a fuller discussion of optional arguments, see Section 8.1 [Setting the language], page 83.

The '`@*`' command begins a *major* or *named section* (corresponding to LaTeX's `\section` command); this command is followed by the section name, terminated by a period. (The period is essential; if it is omitted, weird errors may result.) Major sections are entered in an automatically generated Table of Contents. They are also printed at the top of each output page. If the full section name is too long to so print, one can shorten it with an optional argument, as in

```
@* [INTRO]INTRODUCTION.
```

The command ‘`@*`*n*’ (not illustrated in the above example) begins a major (sub)section of level *n*, where ‘`@*0`’ is equivalent to the simple ‘`@*`’, ‘`@*1`’ indicates a subsection, and ‘`@*2`’ indicates a subsubsection. The highest permissible major level is 2 (a subsubsection). Such subsections are also entered in the Table of Contents. For more information, see Section 10.1.3.4 [Sections], page 95.

As the example demonstrates, the name of the very last section, which should be starred, should be ‘`\INDEX`’. Note the backslash; ‘`\INDEX`’ is a TeX macro. This command tells FWEAVE to write out the index in a special two-column format. By default, ‘`\INDEX`’ expands to ‘`INDEX`’, but this name can be overridden by the style-file parameter ‘`index.name`’ (see Section 12.3.1.1 [S_index], page 113). For more discussion of FWEB's indexing facilities, see Chapter 11 [Index], page 103.

Minor (*unnamed*) sections are begun by ‘`@␣`’(“at-space”); these have no associated names and are not entered into the Table of Contents. A newline counts as a space.

2.2.0.2 The TeX part

All sections begin with (optional) TeX commentary. That can just be straight text; to input that, no knowledge of TeX is required. It can also include mathematical exposition or any of the other advanced features offered by TeX.

Whenever FWEB is in TeX mode, one can temporarily shift into *code mode* by enclosing the code within vertical bars. That code is typeset just like code in the code part (see below), except that newlines are replaced by spaces. Thus, one can say things like

```
Consider the C code fragment ‘|@c for(i=0; i<10; i++){}|’, which ...
```

(If the global language were C instead of FORTRAN, the ‘`@c`’ inside the vertical bars would not be necessary.) The ability to switch back and forth between text mode and code mode at will allows for a very convenient and flexible style of exposition.

2.2.0.3 The definition part

The TeX part 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`’, or ‘`@W`’ (explained later). In the previous example, the first section has a definition part consisting of one FWEB macro definition (‘`@m`’); the second section has no definition part. For more information, see Chapter 7 [Macros], page 62.

(Failure to appreciate how easy it is to shift from part to part can get one into trouble. For example, don't write documentation such as ‘`Consider the @m command`’, because the ‘`@m`’ will inadvertently terminate the documentation part and begin the definition part. What one needs to do here is to use the literal ‘`@`’, as in ‘`@@m`’.)

2.2.0.4 The code 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 8. Within the code part, one can place any sequence of code or code fragments (they need not be complete subroutines) that are valid for the current language. (Setting the language is described in Chapter 8 [Languages], page 83.) The code part is terminated by the next appearance of '`@*`' or '`@␣`'(which signal the beginning of a new section), or by the end of file.

2.2.0.5 The limbo section

The portion of the source file before the first section (i.e., before the first '`@*`' or '`@␣`') is called *in limbo* or *the limbo section*. The only '`@`' commands that are allowed in limbo (in addition to '`@@`', which stands for the character '`@`' and is allowed anywhere) are the language-changing commands, and one of those, such as '`@c`', should appear. Other text in limbo is ignored by FTANGLE and is copied by FWEAVE to the `tex` output file. Thus, one can make or issue TEX macro definitions in limbo that override the defaults in FWEB's macro package '`fwebmac.sty`'. In the above example, see the `\Title` command. This is defined in '`fwebmac.sty`', and basically issues LaTEX's `\title` command.

(Another way of getting TEX text into the limbo section is by means of the '`@l`' command; see Section 5.5.14 [ATl], page 45.)

LaTEX users may need to know that TEX commands in limbo are executed *after* the '`\begin{document}`' command (which is issued automatically in '`fwebmac.sty`'). For more information, see Section 10.1.3 [LaTeX], page 93.

2.3 Modules

The code parts of (possibly noncontiguous) sections can be combined into *modules*. For FWEAVE, this is a *logical* combination, for purposes of cross-referencing different pieces of the code. But for FTANGLE, the combination is physical; FTANGLE's output proceeds module by module.

Modules can be *named* or *unnamed*. There is exactly one unnamed module. The fundamental operation of FTANGLE is that

FTANGLE *outputs the unnamed module.*

That output goes to a compilable file with an extension appropriate to the current language.

The contents of a module, either unnamed or named, consists of a mixture of code and comments. FTANGLE ignores the comments; FWEAVE treats them as TEX text. Within any TEX text, including comments, constructions delimited by '`|...|`' signify a temporary shift into code mode. (In the present design, one cannot enclose a comment within the vertical bars.)

2.3.1 The unnamed module

The unnamed code module is introduced by the command '`@a`'. Subsequent uses of '`@a`' accrete code to the unnamed module. To repeat, the fundamental operation of FTANGLE is that

FTANGLE *outputs the unnamed module.*

Thus, there must be at least one '`@a`' in the source file or FTANGLE will output nothing.

(Why is the command called '`@a`'? Historically, it was the first letter of the alphabet, as befits its prominent status. However, one can also think of it as "accrete.")

2.3.2 Named modules

Named modules represent logically-connected fragments of code.

A module name is specified by the construction

`@<` *Arbitrary TEX text* `@>`

Leading and trailing white space around the name text is ignored. The name text can include the '`|...|`' construction, which tells FWEAVE to typeset a code fragment. Thus, module names can be highly explicit—for example,

```
@< Check that |x >= 0.0|; |abort| if not @>
```

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 module. Thus, a named module might ultimately consist of the code from sections 2, 5, and 9, 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;
}
```

There are several details to notice about the above example. First, FWEAVE considers module names to be simple expressions (such as the single identifier x). In C, expressions are made into complete statements (as is required in the body of a **for** statement) by appending a semicolon. In this case, a *pseudo-semicolon* '`@;`' is appropriate; for more discussion of that, see Section 5.13.2 [AT;], page 58.

Second, after a name has appeared once in full, it may be abbreviated by a unique prefix followed by three periods, 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@>@;
```

Subsequently one can accrete to the above named sections, as often as desired and in any order. This way, definitions of global variables can be introduced anywhere in the `web` source file as logical and pedagogical exposition dictates, 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 subsequent unnamed sections.

Very rarely, one might try the following construction:

```
@
@a
@< Left side @> = @< Right side @>@;
```

Here the intent is to construct an assignment statement. However, this will be flagged as an error because FWEB thinks one is trying to define the named module '`@<`*Left side*`@>`', which one shouldn't be doing while in code mode. To make it work, just put the invisible expression '`@e`' (see Section 5.13 [ATe], page 57) before the equals sign.

2.4 Phases of processing

The FWEB processors perform their work in several distinct phases. (The following is somewhat technical. Scan it, then use it for reference later if necessary.)

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 the source;
- expands FWEB preprocessor commands such as '`@#if`' (see Section 7.3 [Preprocessing], page 80);
- expands '`@'...'`' (see Section 5.6 [ATquote], page 51), '`@"..."`' (see Section 5.6.2 [ATdquote], page 51), and the binary notation '`0b...`' (see Section 8.2.1 [C], page 84) [in FORTRAN, also the octal notation '`0...`' and the hexadecimal notation '`0x...`'];
- stores code text in appropriate modules;
- memorizes macro definitions ('`@d`' and '`@m`') (see Section 5.5.6 [ATd], page 42 and Section 5.5.16 [ATm], page 45).

Phase 2

- outputs outer macro definitions ('`@d`');
- outputs the unnamed module ('`@a`');
- expands FWEB macros ('`@m`');
- expands built-in macros such as '`$IF`' or '`$PI`' (see Section 7.2.3 [Built-in functions], page 66);
- translates RATFOR statements (see Chapter 9 [Ratfor], page 89).

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 phase 2, the source file is read again, then pretty-printed with some cross-reference information. (For discussion of pretty-printing, see Section 10.2 [Pretty-printing], page 100.) In phase 3, an automatically-generated Index, List of Modules, and Table of Contents are written.

More specifically, phase 1

- tokenizes and stores identifiers and module names;
- collects cross-reference information (including, in C and C++, the scanning of '`#include`' files for '`typedef`' and/or '`class`' declarations (see Section 4.2.17 [-H_], page 20);
- stores limbo text definitions made with '`@l`' (see Section 5.5.14 [ATl], page 45);
- collects information about overloaded operators ('`@v`') and identifiers ('`@W`'). See Section 5.5.27 [ATv], page 49 and Section 5.5.28 [ATW_], page 50.

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 it to appropriate TEX macros.

Phase 3 writes out cross-reference information. (To eliminate some of that, see Section 4.2.67 [-x], page 34.) Specifically, it

- writes out the Index ('`INDEX.tex`' by default, but see Section 3.2 [Output files], page 13 and Section 12.3.1 [Index params], page 113);
- writes out a list of named modules ('`MODULES.tex`' by default, but see Section 3.2 [Output files], page 13 and Section 12.3.2 [Module params], page 114);
- writes out macros to generate the Table of Contents. (Table of Contents information is actually processed by LaTEX, not FWEAVE. The information is written to the '`aux`' file.)

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 by system installers in '`custom.h`' (see Chapter 12 [Customization], page 107). The character that initiates the file-name extension (normally a period) can be changed with the '`-E`' command-line option (see Section 4.2.13 [-E_], page 19).

3.1 Input files

FWEB reads files with a variety of default extensions.

'`.fweb`' — Initialization file (optional; for setting up default options used for all runs). This file is always in the user's home directory. See Section 12.2 [Initialization], page 108.

'`fweb.sty`' — Style file (optional; for customizing the behavior of a particular `web` file or group of files). See Section 12.3 [Style], page 112. This file is always in the directory of the `web` file that is being tangled unless that is changed by environment variable `FWEB_STYLE_DIR`. The basic name can be changed by the '`-z`' option (see Section 4.2.71 [-z], page 35). A sample '`fweb.sty`' file is provided with the FWEB distribution.

'*name*`.web`' — Source file.

'*name*`.ch`' — Change file (optional; for making incremental changes to a `web` source file). See Section 3.3 [Change files], page 13.

'*name*`.hweb`' — Code included into web file with '`@i`' (see Section 5.5.9 [ATi], page 43). Include files are searched for in the path set by the environment variable `FWEB_INCLUDES` and/or the '`-I`' option (see Section 4.2.19 [-I_], page 21). If that path is empty, then the current directory is searched.

'*name*`.hch`' — Optional 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 4.2.14 [-e], page 19). When this option is 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., '`ext.web = "web wb"`'; the first one that matches is used.

3.2 Output files

FWEAVE writes a variety of output files.

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

'CONTENTS.tex' — Temporary file that accumulates Table-of-Contents information. (For LaTeX, the 'aux' file is used instead.)

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

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

The names of the three temporary files can be changed with style-file parameters (see Section 12.3 [Style], page 112). Commonly, one may put into the style file 'fweb.sty' commands such as

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

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

FTANGLE writes files of the form

'*name.ext*' — Compilable output 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	C
Fortran–77	suffix.N	f	for
Fortran–90	suffix.N90	f90	for90
Ratfor–77	suffix.R	r	rat
Ratfor–90	suffix.R90	r90	rat90
TeX	suffix.X	sty	sty
VERBATIM	suffix.V	mk	mk

For example, to change the default extension for a C++ file from 'C' to 'c++', put into 'fweb.sty' the line

```
suffix.C = "c++"
```

3.3 Change files

The primary input to the FWEB processors is the 'test.web' source file. 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 an 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. Comments may also be included on the '`@x`', '`@y`', and/or '`@z`' lines.

4 RUNNING FWEB

FWEB has a UNIX-style command-line syntax. There are many command-line options, but few or none of these are necessary for standard appplications. Proceed in blissful ignorance until you need to do something tricky, then scan the list of options to see if they can help.

Commonly-used command-line options can be placed into the initialization file '`.fweb`' (see Section 4.2 [Options], page 15) that resides in one's home directory.

A *style file* (patterned after the utility `makeindex`; see Section 12.3 [Style], page 112) can be associated with each manuscript or collection of related manuscripts in order to customize their appearance. This file is read *after* the command-line options are processed, except that the '`-p`' option gets special treatment; see Section 4.2.46 [-p], page 28.

4.1 Command-line syntax

The command-line syntax is

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

A file name is anything that doesn't begin with a '`-`', except that a lone hyphen stands for the special file name '`stdin`', which means 'read from the standard input.' (This should not be used except for very special effects.)

Command-line options begin with a '`-`'. File names and options can be intermixed, or the options may appear after the file names. The first file name encountered is the web source file; the second, if it exists, is the change file (see Section 3.3 [Change files], page 13). [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 (see Chapter 12 [Customization], page 107), or can be set in the style file '`fweb.sty`'. See Section 12.3.8.15 [null_file], page 120.]

The web file is shown as required since one is normally processing a source. However, some of the information options (see Section 4.2.82 [Info options], page 37) will work without specifying any file name. For example, one can obtain a list of all of the style-file parameters and their default values by saying '`ftangle -Z`'.

4.2 Command-line options

Command-line options may be put, one per line, into the initialization file '`.fweb`' (which is always in the user's home directory). In that file, options beginning with a hyphen are processed *before* the command-line options (so command-line options can override the defaults). To force an option to be processed *after* the command-line options, preface it with an ampersand rather than a hyphen; this is rarely necessary.

To make sense of the plethora of options, it helps to know that options beginning with '`n`' are related to FORTRAN; those beginning with '`r`' are related to RATFOR. Some flags that can be set separately for those two languages also have a global option that sets the flags for both languages simultaneously; cf. '`-n/`', '`-r/`', and '`-/`'.

Some options take arguments. For example, an FWEB macro can be defined from the command line by saying something like '`-mIBMPC=1`'. Unlike many UNIX utilities, *no spaces* *are allowed between any option and its argument.* For example, if one says '`-m IBMPC`', FWEB will think that '`IBMPC`' is a file name.

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 isn't used very often, but it is useful if an option such as '`-v`' is turned on in the '`.fweb`' initialization file and one wishes 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 scrap sequences.

A *scrap* is a part of speech. The expression '`x + y`' consists of three scraps: '`x`' (an expression), '`+`' (a binary operator), and '`y`' (an expression). FWEAVE contains *production rules* such as "replace the combination '`expr binop expr`' with '`expr`'." If all goes well, the result of FWEAVE's reduction process is ultimately just one scrap, such as '`function`'. If FWEAVE is left with more than one scrap at the end of a section, this is called an *irreducible scrap sequence*; '`-1`' displays them.

Irreducible scrap sequences can arise either because the programmer made a mistake or because FWEAVE has not been taught the proper grammar.

While FWEAVE is reducing the scraps, it appends TeX macros that ultimately produce the pretty-printed output. Frequently people ask how to change the appearance of that output. Fundamentally, this is not possible at present; the grammar rules and the associated TeX are hard-coded. A completely general, user-customizable scheme is very complex and daunting; it has not been attempted.

This brief debugging mode can be turned on more locally by means of the '`@1`' command. See Section 5.1.2 [AT1], page 38.

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

This option tells FWEAVE to display detailed reductions of the scraps as it does the pretty-printing. (For a discussion of scraps, see Section 4.2.2 [-1], page 16.) Sometimes FWEAVE fails spectacularly at pretty-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 (usually the system developer!) 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 38.

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

This option supplies information about the '`@`' control codes (see Chapter 5 [AT commands], page 38). It shows the associated style-file parameters that can be used to remap the codes (but *don't do that!*), and it displays the precedence. (Some codes such as '`@@`' may be used anywhere; others such as '`@*`' begin a new section or part of section. 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]'.)

The option produces two columns of output: the first is sorted numerically, the second alphabetically. The notation 'USED_BY_OTHER' means that this command is ignored by whatever processor (FTANGLE or FWEAVE) is currently being run, but may be used by the other processor. (For technical reasons, a very few commands such as '@i' do not show up in this output at present.)

If one says just '-@', information about all control codes is produced. Selected control codes may be queried by listing them after the '-@'. For example, to learn about the commands '@~' and '@a', say '-@~a'. Remember to quote certain characters on UNIX systems—e.g., '-@'*?''. If a command is used by neither processor, its description will be replaced by a question mark.

4.2.5 '-A': Turn on ASCII translations

This option is used primarily for debugging. FWEB works internally with the ASCII character set. If FWEB is run on a non-ASCII machine (notably IBM mainframes), 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 when it encounters 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 (FWEAVE)

Number **do** and **if** blocks in woven FORTRAN and RATFOR output. This feature is particularly useful in FORTRAN-77 to help correlate the beginnings and ends of long blocks (but note that appropriate use of literate programming techniques can keep all of one's blocks short!). Output something like the following is produced, where the comments are inserted automatically by the '-b' option:

```
do i=1,10 // Block 1
 do j=1,10 // Block 2
  if(i==j) then // Block 3
        call sub1(i)
  else // Block 3
        call sub2(i,j)
  endif // Block 3
 end do // Block 2
end do // Block 1
```

The precise form of the block comment that is emitted can be changed by redefining the macro \Wblock in 'fwebmac.sty'.

4.2.8 '-C': Set the color mode

The option '-C*n*' sets the color mode to *n*, where the color modes are, briefly,

0 No color

1 ANSI color

2 Bilevel

3 Trilevel

4 User-defined

These modes, and color output in general, are described more thoroughly in Section 12.3.7 [Color], page 117.

For obscure technical reasons, this command is processed differently than all other command-line options. In the present incomplete implementation, *the color mode must be set on the command line*, not in '.fweb'! To work around this annoyance, UNIX users could alias commands such as 'ftangle -C1'.

4.2.9 '-c': Set global language to C

Usually the global language (Chapter 8 [Languages], page 83) is set to C by means of the command '@c' in limbo, rather than using '-c' on the command line. However, one may need to use the command-line option '-c' if a subsequent command-line option is language-dependent. See, for example, the discussion of the option '-D' in Section 4.2.11 [-D_], page 18.

4.2.10 '-c++': Set global language to C++

For more information, see the discussion of '-c' in Section 4.2.9 [-c], page 18.

4.2.11 '-D': Display reserved words

This information option displays the list of reserved words for the language currently in force. (For the purposes of this option, 'reserved words' include "true" reserved words such as 'int'; they also include the names of intrinsic functions such as 'sin' and, for FORTRAN and RATFOR, I/O keywords such as 'IOSTAT'.) 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 one says '-Dabc', one will get just the reserved words that begin with "abc".

If one says '-D*', one will get all reserved words for all languages.

The '-D' may be followed by a list of one or more optional letters enclosed in square brackets. (For UNIX systems, don't forget to quote the brackets, as they mean something special to the shell.) The letters represent which kind of reserved word to display; they may be 'i' ('intrinsic'), 'k' ('keyword'), or 'r' ('reserved'). Thus, to see a list of the FORTRAN keywords, say '-D[k]'. To see a list of the intrinsic functions for C++ that begin with 's', say '-Lc++ -D[i]s'.

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

(This option is obsolete.)

4.2.13 '-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,'. This feature is required by at least one perverse system.

4.2.14 '-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 one intends. 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., 'ext.web = "web wb"'; the first one that matches is used.

4.2.15 '-F': Compare output files with old versions (FTANGLE)

When the '-F' option is in effect, FTANGLE writes its output to a temporary file (or files) instead of to its ultimate destination such as '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 a make file won't recompile the output unless it really has to.

Note that with this option in effect, if one uses 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. This is harmless, but annoying. To get things back in sync, do a run without the '-F'.

The location of the temporary file as well as details of the renaming procedure are determined by the automatic configuration script ./configure during installation of the processors. The script 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. That 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.

Some of the above-mentioned file names and system commands are system-dependent; see Chapter 12 [Customization], page 107.

4.2.16 '-f': Turn off module references for identifiers (FWEAVE)

In an attempt to be helpful, FWEAVE appends subscripts to many identifiers indicating in which section they are first defined (see Section 12.3.4 [Subscript params], page 114). Sometimes these result in output that is too cluttered and confusing. The '`-f`' option turns off the subscripting operations.

4.2.17 '-H': Scan C/C++ include files (FWEAVE)

For C or C++, the '`-H`' option tells FWEAVE to do a phase-1 scan of `#include` files for '`typedef`' and/or '`class`' declarations. This removes the necessity of including many redundant '`@f`' format statements (see Section 5.5.8 [ATf], page 42), which would otherwise be necessary in order that the code be pretty-printed correctly. For example, if one uses the '`-H`' option with the code

```
@c++
@
#include <Complex.h>
Complex z;
```

the identifier **Complex** will be properly formatted as a reserved word (in boldface), as though one had said '`@f Complex int`'.

In addition to the basic '`-H`', there are several more detailed options:

`-Hx` Make index entries only for double-quoted include files.

`-HX` Make index entries for all include files.

`-Hr` Retain temporary files generated by the preprocessor.

By default, index entries are not made for variables that are read during such scans. If one says '`-Hx`', index entries will be made only for include files whose names are enclosed in double quotes rather than angle brackets, such as '`#include "myheader.h"`' (usually these are defined by the user and reside in the local directory). If one says '`-HX`', index entries will be made for all include files. This can generate many entries, since system header files may be complicated and may include other files as well.

This command is implemented as follows. When FWEAVE reads an `#include` statement, it issues a `system` command to run the C preprocessor on the included file. Output from the preprocessor is written to a temporary file, which FWEAVE scans.

By default, the C preprocessor will look in certain default paths for the included files. To add to those defaults, use one or more '`-I`' options *after* the '`-H`'. These colon-delimited lists are concatenated to the contents of the environment variable `FWEB_HDR_INCLUDES`, if that is defined. The entire list is then passed as multiple '`-I`' options to the preprocessor.

This command, new with version 1.53, is *highly experimental* and incomplete. The installation script attempts to determine what command to use to run the preprocessor, but that is not guaranteed to work in general. '`-H`' has been tested only with `gcc`.

To send arguments to the C preprocessor, see Section 4.2.65.4 [-WH_], page 33.

The '`-H`' mechanism uses temporary files to do its work. By default, those are deleted after use. However, for debugging purposes, one can force those to be retained by saying '`-Hr`'. That option also has the side effect of displaying the actual command line that was sent to the preprocessor.

4.2.18 '`-h`': Get help

If just '`-h`' is typed, a message is printed saying where further help is available. It refers one to the various information options (see Section 4.2.82 [Info options], page 37) and the on-line documentation (see Chapter 15 [Support], page 131). If the stand-alone `info` program (the GNU hypertext browser) is installed, one can enter '`info FWEB`' at this time by typing '`?`' or a space-separated list of FWEB menu items such as '`Macros FWEB built-in $PI`'. In fact, since '`$PI`' appears in the detailed node listing, one can simply type '`$PI`'. More generally, one can type anything that `info` accepts on its command line (the option '`-f FWEB`' is implicit).

One can bypass the printed message and directly enter `info` by specifying the `info` arguments as arguments to '`-h`'. For example, on a UNIX system, one could type '`-h'\$PI'`'. Here the dollar sign must be escaped because it has special significance to the shell, and the quotes are necessary in order to preserve that escape character as the argument is supplied to `info`. To get to the top-level FWEB info directory, type '`-h.`' or '`-h'?'`'.

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

The fundamental search list for *include files* (read in via '`@i`' or '`@I`') 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.

For information about include files, see Section 5.5.9 [ATi], page 43.

4.2.20 '`-i`': Don't print '`@I`' include files (FWEAVE)

If a web file is included via '`@I`' (see Section 5.5.10 [ATI_], page 44), 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 of the time. (C and C++ programmers will find that the '`-H`' option substantially reduces the need to include such header files; see Section 4.2.17 [-H_], page 20.)

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

By default, identifiers that are referenced in non-printed include files are not cross-referenced or indexed in any way. To force them to be cross-referenced, say '-ix' instead of '-i'. In the present implementation, the cross-reference information for such non-printed files is presented in the form '#*n*', where *n* is the integer section number. (The LaTeX section label is undefined for sections in non-printed files.)

The option '-i!' means skip the include files completely. This is usually not very useful.

4.2.21 '-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 4.2.20 [-i], page 21) is more often used.

4.2.22 '-j': Inhibit multiple includes

File inclusion via FWEB's '@i' command suffers from a design deficiency: they cannot be inhibited by means of FWEB's preprocessor commands. (The reason is that '@i' is processed very early in the input stage, before tokenization. This design decision was inherited from CWEB, and is very difficult to change.) A particularly annoying situation arises when the same file is included multiple times; various array space may be eaten up unnecessarily. The '-j' option inhibits such multiple includes.

4.2.23 '-k': Don't recognize lower-case forms of keywords

By definition, in FORTRAN and RATFOR, a keyword is one of the parameters such as `IOSTAT` used in the parameter list of an I/O statement. For example,

```
open(21, FILE=file_name, STATUS='old', IOSTAT=io_flag)
```

Such keywords are typeset in `typewriter type` to better highlight them. In FORTRAN, these keywords are case-insensitive. However, note that certain of the lower-case forms—in particular, 'end', 'read', and 'write'—have other special meanings, and one can in principle use any of these keywords as ordinary variables in other parts of the code; however, FWEB identifiers can have just one meaning throughout the code. By default, the lower-case forms are also recognized as keywords (except for the three special identifiers just mentioned), so one shouldn't use those as regular variables. To cause only the upper-case forms to be recognized, use the '-k' option.

4.2.24 '-L': Select global language

To select a global language from the command line, say '-L*l*', where *l* is one of `{c,c++,n,n9,r,r9,v,x}`. See Chapter 8 [Languages], page 83.

Usually, the global language is set via an '@' command in limbo, not on the command line. However, one may need to use a command-line option such as '-L_' if a subsequent command-line option is language-dependent. See, for example, the discussion of the option '-D' in Section 4.2.11 [-D_], page 18.

4.2.25 '-l': Echo input line

The option '-l*[mmm[:nnn]]*' echoes the input lines 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 (usually by the system developer). 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.

4.2.26 '-M': Set output message level

By default, FWEB is relatively verbose; as it proceeds, it prints messages about what files it is reading and writing, numbers of the starred sections, line numbers, etc. However, different levels of verbosity can be set by the command '-M*level*', where the level may be 0 (least verbose) through 4 (most verbose; the default), as described in the following table:

0 Like level 1, but the start-up banner is not printed. If FWEB runs to completion with no errors, nothing at all will be printed.

1 Print only error messages.

2 Print error and warning messages.

3 Print errors, warnings, and short information messages (excluding starred section numbers and line numbers).

4 Print everything.

The start-up banner, which includes the version number, is printed for all message levels except 0. For level 0, one can use the '-V' option to request the start-up banner. See Section 4.2.63 [-V_], page 32.

This option is very recent, and may not be fully debugged for obscure combinations of command-line options. Please report any annoyances.

Another way of discriminating message types is via color output. See Section 12.3.7 [Color], page 117.

4.2.27 '-m': Define FWEB macro (FTANGLE)

The command-line construction

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

defines the FWEB macro `A` as though the definition

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

had appeared in the first definition part of the web file.

One can also say '-m'A(x) x'', where the quotes are removed by the shell. That is, an '=' appearing *immediately* after the macro name (or argument list, if there is one) plays the role of the space in the conventional definition. Thus, carefully distinguish the forms

```
-m'A(x)=x'    // A(x) expands to 'x'
-m'A(x) =x'   // A(x) expands to '=x'
-m'A(x)==x'   // Precisely equivalent to the previous example.
```

The equals sign is permitted only with command-line macro definitions, not with '@m' commands (see Section 5.5.16 [ATm], page 45) in the definition parts of the web file.

4.2.28 '-m4': Understand m4 built-in commands

This tells FWEAVE to properly format the reserved words of the `m4` preprocessor. The use of that preprocessor is *not recommended* in conjunction with FWEB; use FWEB's built-in C-like preprocessor instead.

4.2.29 '-m;': Append pseudo-semicolons

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

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

```
@m SET(x,y) x=1; y=2@;
@m TEST(x) if(x) y; else z@;
```

4.2.30 '-n': Set global language to FORTRAN–77

This is FWEB's default, so it generally does not need to be used explicitly. (See also the discussion of Section 4.2.24 [-L_], page 22.) However, variants of this option, as described below, may be useful.

See also Chapter 8 [Languages], page 83 and Section 8.2.3 [Fortran], page 85.

4.2.31 '-n9': Set global language to FORTRAN–90

See Chapter 8 [Languages], page 83 and Section 8.2.3 [Fortran], page 85; see also the discussion of '`-L`' in Section 4.2.24 [-L_], page 22.

4.2.32 '-n@;': Supply automatic pseudo-semicolons [FORTRAN]

(Don't forget that a semicolon has special meaning to UNIX shells, so you'll probably have to quote this command: '`-n'@;''`'.)

This is the default mode of operation for free-form FORTRAN-90; the input driver automatically appends a pseudo-semicolon (invisible) to each logical line of source code. Since it is the default, one doesn't have to use it unless one wishes to negate it (see Section 4.2.1 [Negating options], page 16). In that case, it is best to place the '`--n@;`' command in the source file, as '`@n9[--n@;]`'. If one places it on the command line, be sure to set the language first: `-n9 --n@;`.

For free-format FORTRAN-90, when '`-n@;`' is in effect (the default), '`-np`' is also turned on. See Section 4.2.37 [-np], page 26.

For further discussion, see the companion command Section 4.2.33 [-n;], page 24.

4.2.33 '-n;': Supply automatic semicolons [FORTRAN]

(Don't forget that a semicolon has special meaning to UNIX shells, so you'll probably have to quote this command: '`-n';''`'.)

This command functions the same as '`-n@;`' (see Section 4.2.32 [-nAT;], page 24, except that actual (visible) semicolons rather than pseudo-semicolons are appended. This is the default mode of operation for FORTRAN-77 (and for that language, it cannot be turned off by negation).

The distinction between '-n@;' and '-n;' has to do with what is visible on output. In FORTRAN-77, semicolons are not printed by default since that seemed to annoy many users. However, that causes trouble with FORTRAN-90 code containing multiple statements per line, as in

```
a = b; c = d
```

If '-np' is not used, then the semicolon in the above example is not printed, hindering legibility. Thus, the default mode of operation for free-format FORTRAN-90 is '-n@;' and '-np'. This turns the above example into 'a = b; c = d@;' and displays it correctly.

When '-n;' is used, semicolons will not be printed by default. To force them to be printed, use the '-np' option (see Section 4.2.37 [-np], page 26).

Do not insert semicolons by hand in FORTRAN-77; they are always inserted automatically. If you have terminated FORTRAN-90 statements by hand, turn off auto-semis by '-n;' (and use '-np' at your discretion).

The following table summarizes the defaults for auto-semi insertion and semicolon printing in FORTRAN, both fixed and free formats ('N/A' means 'not applicable'):

	Fixed	Free
F77	'-n;'	N/A
F90	'-n;'	'-n@; -np'

4.2.34 '-n:': Put statement label on separate line [FORTRAN]

By default, in FORTRAN statement labels are placed on the same line, and backspaced from, the command that is being labeled, as in

```
EXIT: continue
```

This can look ugly if the label is very long. The command '-n:' places the label on a separate line, as is done automatically for RATFOR—e.g.,

```
EXIT:
  continue
```

If neither of these options appeals to you, you could try redefining the macro \Wlbl, found with some discussion in 'fwebmac.web'. That macro is emitted only when '-n:' is *not* used.

4.2.35 '-nb': Number ifs and dos [FORTRAN] (FWEAVE)

In the woven output, extra comments are added to help one correlate the block structure of the code. For more discussion, see Section 4.2.7 [-b], page 17.

4.2.36 '-nC': Ignore single-line comments [FORTRAN]

Ignore, at the input-driver stage, comment lines beginning with 'C', 'c', or '*'.

Interpretation: In the usual mode of operation, the FORTRAN-77 input driver makes a heroic attempt to mix the original single-line column-1 commenting style with the FWEB style ('/*...*/' and '//'). It converts single-line comments to the '/*...*/' style and passes them along to the innards of the processors.

Problems sometimes arise when converting an existing FORTRAN code to FWEB. Such codes may have very large blocks of code or documentation commented out with a 'C' in column 1. Special TEX characters in those comments can cause problems for FWEAVE; sometimes FTANGLE gets confused as well. The '-nC' option short-circuits these problems by simply throwing all such lines away at the input driver stage.

This option is not a recommended long-term solution. Instead, consider the following:

- In FWEB, blocks of code should be commented out with the preprocessor commands `@#if 0...@#endif`; see Section 6.3 [Temporary comments], page 61.
- Textual comments for documentation should be converted to the standard FWEB commenting style.
- If one has a block of code prefaced by an extremely long comment, replace that by a named module and put the comment into the TEX part of that section.

4.2.37 '-np': Print semicolons [FORTRAN] (FWEAVE)

Although the FORTRAN input driver automatically terminates logical lines with semicolons (FORTRAN-77; see Section 4.2.33 [-n;], page 24) or pseudo-semicolons (FORTRAN-90; see Section 4.2.32 [-nAT;], page 24) so that the innards of FWEAVE can process them correctly, the semicolons are not printed by default. To make actual semicolons be printed, use the '-np' option.

'-np' is turned on automatically for free-format FORTRAN-90 when '-n@;' is in effect (the default). For more discussion, see Section 4.2.33 [-n;], page 24.

4.2.38 '-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 (as it would be in C). 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.

See also Section 4.2.39 [-n&], page 26.

4.2.39 '-n&': Free-form syntax continued by ampersand

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

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

For FORTRAN-90, free-form syntax continued by the ampersand is FWEB's default, so one probably will not need to use '-n&' explicitly.

See also Section 4.2.38 [-n\], page 26.

4.2.40 '-n/': Recognize short comments [FORTRAN]

The standard FWEB notation for a short comment (one terminated by the next newline) is '// ...'. However, in FORTRAN the '//' denotes concatenation by default. To make it denote a short comment, use the '-n/' option. One can do this in the '.fweb' file (see Chapter 12 [Customization], page 107) or with the language-setting command in limbo, as in '@n/'.

In FWEB, one may always use '\/' for concatenation, so there's no penalty for using '-n/'.

4.2.41 '-n!': Make '!' denote short comment [FORTRAN]

In FORTRAN-90, '!' starts a short comment. However, by default FWEB usurps '!' for the logical not, as in 'if(x != y)'. To force it to recognize '!' as a comment, use '-n!'. However, the recommended style is to use FWEB's standard convention that '//' denotes the start of a short comment (see Section 4.2.40 [-n/], page 27). See also Section 4.2.81 [-!], page 37 and Section 4.2.56 [-r!], page 30.

In FORTRAN-77, to include the exclamation point inside a string, escape it with a backslash, as in

```
s = "A \! inside a string"
```

This possibly annoying restriction arises because the unduly complicated FORTRAN input driver does some preprocessing of the FORTRAN source before it feeds it to the cores of the processors.

4.2.42 '-n)': Reverse array indices [FORTRAN] (FTANGLE)

This *somewhat experimental* flag permits FORTRAN programmers to use C-style array indices. Conversions such as the following are made (during the output phase of FTANGLE):

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

[No spaces may intervene between ')' and '('; effectively, ')(' is treated as one token for the purposes of '-n)'.] This feature permits convenient definitions of macros that deal with multi-dimensional vectors.

Unfortunately, FTANGLE doesn't fully understand the syntax of the source code—and never will, unless it is fully integrated with a compiler. It will therefore be confused by situations like the following FORTRAN example:

```
dimension x(0:4)(1:2) // OK
character*90 ch(4) // OK
write(6,*) ((x(i)(j),i=1,2), j=3,4) // Will reverse incorrectly.
c = ch(4)(3:4) // Shouldn't reverse, but will.
```

One solution, due to Charles Karney, is to insert a space to prevent ')(' from being recognized as a single token. However, since ordinary white space is eaten on input, one must resort to something like the following ('$UNQUOTE' is a built-in FWEB function; see Section 7.2.3.64 [$UNQUOTE], page 78):

```
@m SP $UNQUOTE(' ')
@a
dimension x(0:4)(1:2)
character*90 ch(4)
write(6,*) SP ((x(i)(j),i=1,2), j=3,4)
c = ch(4)SP(3:4)
```

This option is controlled by the three style-file parameters 'paren.len', 'paren.num', and 'paren.nest'. (See Section 12.3 [Style], page 112.) 'paren.len' is the default number of bytes to be allocated for each index; if an index is longer than this number, the current length is increased by this number and storage is automatically reallocated. 'paren.num' is the maximum number of allowed indices; for example, when processing 'a(i)(j)(k)', 'paren.num' is 3. 'paren.nest' is the maximum parenthesis nesting level. In the example 'x(a(i)(j)(k))', 'paren.nest' is 2. If either of the last two parameters is exceeded, a message will be issued asking you to increase the appropriate value.

4.2.43 '-o': Don't overload operators

This option inhibits the operator-overloading feature invoked by the command '@v' (see Section 10.2.3 [Overloading], page 101).

4.2.44 '-q': Don't translate RATFOR

(This option is obsolete.)

4.2.45 '-P': Select TeX processor

Say '-PT' or '-PL' to inform FWEAVE that its output will be processed by TeX or LaTeX, respectively. Beginning with Version 1.50, the default processor is LaTeX ('-PL'). If you always use TeX, it's easiest to put '-PT' into the '.fweb' initialization file.

Please note that '-PT' is no longer supported; FWEB *development is now based exclusively on LaTeX.*

4.2.46 '-p': Buffer up a style-file entry

This option specifies a style-file entry (see Section 12.3 [Style], page 112). Its argument is exactly the same as a line that one may put into the local FWEB style file. Thus, if in '`fweb.sty`' one would say '`entry="value"`', the form of the '`-p`' option would be '`-pentry='"value"'`'. (The single quotes are required on a UNIX system because the double quotes have special significance to the shell.)

This option can be used either in the '`.fweb`' initialization file (see Section 12.2 [Initialization], page 108), to record style-file entries that are common to all runs, or on the command line, to override a local style-file entry for a single run. This behavior is a consequence of the following order of processing style parameters:

1. '`-p`' options in '`.fweb`';
2. entries in the local style file '`fweb.sty`';
3. '`-p`' options on the command line.

4.2.47 '`-r`': Set global language to RATFOR–77

See Chapter 8 [Languages], page 83 and Chapter 9 [Ratfor], page 89. See also Section 4.2.24 [-L_], page 22.

4.2.48 '`-r9`': Set global language to RATFOR–90

See Chapter 8 [Languages], page 83 and Chapter 9 [Ratfor], page 89. See also Section 4.2.24 [-L_], page 22.

4.2.49 '`-rg`': Set goto parameters

This obscure option is used for configuring RATFOR (and really should be a style-file parameter). (Discussion not finished.)

4.2.50 '`-rk`': Suppress comments about RATFOR translation (FTANGLE)

By default, the RATFOR translator writes comments about what command it is translating. The '`-rk`' option suppresses those comments. Arguments to this option allows one to suppress comments about only particular commands, according to the following list:

`b` — `break`
`c` — `case`
`t` — `default`
`d` — `do`
`f` — `for`
`i` — `if`
`n` — `next`
`p` — `repeat, until`
`r` — `return`
`s` — `switch`
`h` — `where`
`w` — `while`

For example, one can say '`-rkrb`' to suppress comments about the **return** and **break** statements.

4.2.51 '-rK': Write comments about RATFOR translation (FTANGLE)

This is the negative of '-rk' (see Section 4.2.50 [-rk], page 29); it forces comments about particular RATFOR commands.

4.2.52 '-r@;': Turn on auto-semi mode using pseudo-semis [RATFOR]

Please don't use this option (it may not work). Insert semicolons by hand in your RATFOR code, just as one does in C.

4.2.53 '-r;': Turn on auto-semi mode using actual semis [RATFOR]

Please don't use this option (it may not work). Insert semicolons by hand in your RATFOR code, just as one does in C.

4.2.54 '-rb': Number ifs and dos [RATFOR]

In the woven output, extra comments are added to help one correlate the block structure of the code. For more discussion, see Section 4.2.7 [-b], page 17.

4.2.55 '-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 '\/'.

For an example, see Section 4.2.40 [-n/], page 27.

4.2.56 '-r!': Make '!' denote short comment [RATFOR]

See the corresponding discussion of '-!' in Section 4.2.81 [-!], page 37 and Section 4.2.41 [-n!], page 27.

In FORTRAN-77, to include the exclamation point inside a string, escape it with a backslash, as in

```
s = "A \! inside a string"
```

4.2.57 '-r)': Reverse array indices [RATFOR] (FTANGLE)

See the corresponding discussion of '-n)' in Section 4.2.42 [-n)], page 27.

4.2.58 '-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. '-sm*nnn*' displays allocations of nnn bytes or more; if *nnn* is missing, 10000 is assumed.

4.2.59 '-T': Flag-setting options for FTANGLE

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

4.2.59.1 '-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 (then be prepared to make some obscure programming errors).

4.2.59.2 '-Tb': Permit built-functions to be redefined

By default, built-in functions such as `$IF` (see Section 7.2.3 [Built-in functions], page 66) may not be redefined by an `@m` command. To allow this extremely dangerous operation, use the '-Tb' option.

4.2.59.3 '-Tm': Permit user macros to be redefined

By default, user macros may not be redefined by an `@m` command. To permit this, use the '-Tm' option. Note that many functions described under Section 7.2.3 [Built-in functions], page 66, such as `$PI`, are in fact implemented as macros.

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

By default, FTANGLE attempts to be helpful and writes 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 (see Section 4.2.15 [-F_], page 19) is in effect, since the header information includes a time stamp that would defeat a successful file comparison.]

4.2.59.5 '-T%': Don't retain trailing comments (TeX)

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, TeX comments that do not begin a line are always retained unless the '-T%' option is used. This option has no effect for languages other than TeX.

4.2.59.6 '-T#': Don't insert '#line' command after '@%'

If the '@%' command (see Section 5.8.3 [AT%], page 52) is used to comment out a line, it eats the trailing newline. An undesirable consequence of this is that, if nothing is done, the subsequent line numbering will be misunderstood by a debugger, at least until FWEB inserts a '#line' command for some reason. To prevent this, FWEB inserts by default an implicit '@#line' command (see Section 7.3 [Preprocessing], page 80) after each '@%' that begins a line. To prevent this from happening (possibly because the feature doesn't work correctly, in which case you should report it; see Chapter 15 [Support], page 131), use the '-T#' option.

4.2.60 '-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-77, 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.

4.2.61 '-U': Convert reserved output tokens to lower case (FTANGLE)

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

4.2.62 '-u': Undefine FWEB macro (FTANGLE)

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

CAUTION: This option 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.

4.2.63 '-V': Print FWEB version number

This flag requests the startup banner, which includes the FWEB version number, to be printed. This is usually done anyway, so it is only relevant when the message level is 0 (see Section 4.2.26 [-M_], page 23).

4.2.64 '-v': Make all comments verbatim (FTANGLE)

By default, comments are not passed to the tangled output. With '-v', all comments are included verbatim in the tangled output. Since there's generally no harm in this, one might want to put this option into '.fweb' (see Section 12.2 [Initialization], page 108).

4.2.65 '-W': Flag-setting options for FWEAVE

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

4.2.65.1 '-W@': Set module warning flag.

FWEAVE can check module names for the possible anomalous conditions of "never used" or "multiple uses." These correspond to a module warning level, as in the following numbered list:

1. Never used.
2. Multiple uses.

The module warning flag is the bitwise OR of the desired warning levels; warning messages are printed only when the relevant bits are turned on. By default, it is 1, so only messages about never-used modules are printed. The '-W@*flag*' overrides the default. For example, to get messages only about multiple uses, say '-W@2'; to get no messages, say '-W@0'. One can put such an option into the '.fweb' initialization file (see Section 12.2 [Initialization], page 108).

FWEAVE will always complain about module names that are never defined.

4.2.65.2 '-W1': Cross-reference single-character identifiers

By default, FWEB does not index uses of single-character identifiers (following Knuth's original design). (It does index their definitions.) To get complete cross-reference information for single-character identifiers, use the '-W1' option.

4.2.65.3 '-W[': Process bracketed array indices

This *experimental* option makes square brackets behave like parentheses in the context of array indices.

In FORTRAN, FTANGLE will just replace the brackets by parentheses. In C, the brackets will be left alone.

FWEAVE, however, will typeset the indices according to the 'fwebmac.sty' macro '\WARRAY'. This macro takes one argument, which is just the array index or indices. (In C, indexing like 'a[i][j][k]' generates the argument 'i,j,k'.) By default, '\WARRAY' just surrounds its argument with brackets. However, the user may change its definition to get special effects such as superscripted or subscripted indices. A simple example macro '\WSUB' is provided in 'fwebmac.sty'; one can say '\let\WARRAY\WSUB' in the limbo section to have bracketed indices print as subscripts.

This feature may not work when the contents of the brackets are too complicated (so that FWEAVE tries to typeset them by going in and out of math mode).

For more information, experts can see 'fwebmac.web', command \WXA.

4.2.65.4 '-WH': Send additional arguments to the C preprocessor

When the '-H' option (see Section 4.2.17 [-H_], page 20) is used, the C preprocessor is invoked to scan include header files for **typedef**'s and **class** declarations. That is called with a standard set of options. (Presently, gcc is actually called to invoke the preprocessor; it is sent the options '-E', '-P', and '-I'.) Occasionally it may be necessary to send additional options. Those can be specified as the (string) argument to '-WH'. Thus, to define two macros to the preprocessor, one could say either of

```
-WH-Dtest1=1 -WH-Dtest2=2
-WH"-Dtest1=1 -Dtest2=2"
```

The first form shows that '-WH' accretes to earlier uses. The second form shows how to handle embedded blanks (in a UNIX shell). Then, if one were programming in C, use of '-H' would issue the system command

```
gcc -E -P -Dtest1=1 -Dtest2=2
```

4.2.65.5 '-WdfFlmvw': Don't print various things in woven output

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

-Wd — outer definitions ('@d' or '@D')
-Wf — format statements ('@f')
-WF — format statements ('@F')
-Wl — limbo text definitions ('@l')
-Wm — FWEB macro definitions ('@m' or '@M')
-Wv — operator overloads ('@v')
-Ww — identifier overloads ('@w' or '@W')

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

4.2.66 '-w': Change name of macro package (FWEAVE)

The option '-w' means "Don't print '\input fwebmac.sty' as the first line of the '.tex' output file." The option '-w*fname*' means "Print '\input fname' as the first line." For example, when working with REVTEX (see Section 10.1.3.2 [REVTeX], page 94), one needs to say '-wrwebmac.sty'.

This option can be used for special effects when one is trying to obtain behavior different from that defined by FWEB's macro package 'fwebmac.sty' (see Section 10.1.2 [fwebmac.sty], page 92). However, try to not do that. Please submit requests for such behavior modifications to the developer; see Chapter 15 [Support], page 131.

4.2.67 '-x': Eliminate or reduce cross-reference information (FWEAVE).

Cross-reference information (for FWEAVE) includes the Table of Contents ('c'), the Index ('i'), and the Module List ('m'). The option '-x' eliminates all of that information. The option '-x*letters*' eliminates the piece of information corresponding to each letter in the list. For example, '-xim' eliminates the Index and the Module List.

Another possibility is to say '-xu', which prevents cross-references from unnamed sections (begun with '@a' or '@A') from appearing in the Index.

4.2.68 '-X': Print selected cross-reference information (FWEAVE)

When used with any of the arguments 'cim', this option is the opposite of '-x'. See Section 4.2.67 [-x], page 34.

The option '-XI' tells FWEAVE to write its index cross-references to a file formatted for input by the makeindex utility. This feature facilitates creation of a master index that spans several individual web files. For more discussion, see Section 11.2 [Using makeindex], page 103.

The construction '-XI' stands alone; one may not mix the 'I' with the list 'cim'. Also, this option is overridden by '-xi', which suppresses output of all index information.

4.2.69 '-y': Allocate dynamic memory

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

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

When FWEB runs out of space, it usually (but not always) issues a message telling one which '`-y`' command to use in order to increase the allocations. (Someday it will reallocate automatically.) One may wish to add some such options to the '`.fweb`' file.

For a more detailed discussion of memory allocation and a menu of the various dynamic arrays, see Section 12.2.2 [Memory allocation], page 108.

4.2.70 '-Z': Display default style-file parameters

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

After printing the defaults, the '`-p`' options (see Section 4.2.46 [-p], page 28) and the style file '`fweb.sty`' are processed. If that processing has overridden any of the defaults, the parameters are printed again, preceded by an asterisk.

To see only the parameters that have been modified from the defaults, say '`--Z`'.

The '`-Z`' option behaves slightly differently for color escape sequences than for other parameters; see Section 12.3.7 [Color], page 117.

4.2.71 '-z': Change name of style file

The command '`-znew.sty`' changes the default style-file name '`fweb.sty`' to '`new.sty`'. The command '`-z`' (with no argument) means "Don't read any style file."

Normally the style file is read from the same directory in which the `web` source file resides (or from the path defined by the environment variable `FWEB_STYLE_DIR`). To force `fweb.sty` to be read from the current directory, say '`-z.`'.

4.2.72 '-.': 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 use FWEB conventions such as '`<`' instead of '`.LT.`'.

4.2.73 '-\': Explicitly escape continued strings

In FWEB, long strings are continued with the backslash. Normally, the continuation of the string must start in the first column of the next line; otherwise, spurious blanks will be introduced. However, when the '`-\`' option is in effect, FWEB expects that the continuation will also begin with the backslash, and it will ignore leading white space and the backslash. (This feature was inspired by FORTRAN-90.) Thus, in the example

```
"This is \
      \continued";
```

the effective string is "This is continued" when '-\' is in effect.

Note that this option affects all strings in the source file; one cannot mix and match.

4.2.74 '-(': Continue parenthesized strings with backslashes

This option is like '-\' (see Section 4.2.73 [-\], page 35), but it refers to certain strings that are not normally quoted, such as the arguments of 'ifelse' commands in m4.

4.2.75 '-:': Set starting automatic statement number

This option is useful for FORTRAN and RATFOR. Symbolic statement labels that are defined with the '#:0' macro command (Section 7.2.2 [Tokens], page 65; Section 8.2.3 [Fortran], page 85), as in '@m EXIT #:0', are incremented starting with the default of 90000. To change this to, e.g., 789, say '-:789'.

4.2.76 '->': Redirect output (FTANGLE)

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

This command has no effect for FWEAVE.

Although the appearance of this command is highly intuitive, it may be hard to type quickly. An equivalent command is '-=' (see Section 4.2.77 [-=], page 36).

4.2.77 '-=': Redirect output (FTANGLE)

Equivalent to '->' (see Section 4.2.76 [->], page 36), and faster to type on many keyboards.

4.2.78 '-#': Turn off comments about line and section numbers (FTANGLE)

By default, tangled output includes comments about the line and section numbers corresponding to the current piece of code. To eliminate this clutter, say '-#'. (But note that the line-number information is very useful for debugging in C and C++, as it enables the debugger to display the source line in the web file.)

In some cases, bugs in tangled output, particularly from FORTRAN, can be eliminated by using '-#'. (But please report the bug anyway; Chapter 15 [Support], page 131.)

In some cases, it is useful to turn off the line- and section-number information locally. This can be done with the '@q' command. See Section 5.5.22 [ATq], page 48.

4.2.79 '-+': Don't interpret compound assignment operators

Both RATFOR and FORTRAN attempt to translate the commands '++', '--', '+=', '-=', '*=', and '/=' into code that behaves as their C/C++ counterparts. To turn this feature off, use '-+'.

Notice that in FORTRAN-90 '/=' is a token for "not equal," so if you want to use that you must turn off the compound assignment operators with use '-+'. However, a better solution is to leave them turned on and use FWEB's standard '!=' token for "not equal."

See also Section 12.2.2.16 [-ylx], page 110.

4.2.80 '-/': Recognize short comments (FORTRAN & RATFOR)

If this command is not used with the FORTRAN-like languages, the '//' construction will be interpreted as concatenation rather than as the beginning of a short comment.

Concatenation can be signified with FWEB's token'\/', so no penalty is incurred for using '-/'.

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'.

See also Section 4.2.40 [-n/], page 27 and Section 4.2.55 [-r/], page 30.

4.2.81 '-!': Make '!' denote short comment (FORTRAN & RATFOR)

This option is not recommended; use FWEB's standard '//' to begin short comments.

To include the exclamation point inside a string, escape it with a backslash, as in

```
s = "A \! inside a string"
```

4.2.82 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 16.

'-D' displays information about reserved words. See Section 4.2.11 [-D_], page 18.

'-y' displays default dynamic memory allocations. See Section 4.2.69 [-y], page 35.

'-Z' displays default values of style-file parameters. See Section 4.2.70 [-Z_], page 35.

The '-h' option reminds one about these information options; it also provides convenient access to the GNU info browser. See Section 4.2.18 [-h], page 21.

5 FWEB COMMANDS

All FWEB 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 13), or column-oriented FORTRAN-77 processing].

Some of these control codes may be used anywhere; others begin a new part of the current section. (For a discussion of sections and parts, see Section 2.2 [Structure], page 5.) For a quick summary of the control-code mappings and to see which codes begin new parts, say '`ftangle -@`'. See Section 4.2.4 [-AT], page 16.

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 38 and Section 5.1.3 [AT2], page 38). The '`@0`' command should appear in a different section from the '`@1`' or '`@2`' commands.

5.1.2 '`@1`': Display irreducible scraps

This is a local version of the command-line option '`-1`' (see Section 4.2.2 [-1], page 16); refer to that discussion for more information.

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

This is a local version of the command-line option '`-2`' (see Section 4.2.3 [-2], page 16); refer to that discussion for more information.

5.2 Literal control characters

Several commands insert specific characters.

5.2.1 '`@@`': The character '`@`'

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

Don't forget to double the '`@`' even inside strings. For example, the FWEB source line

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

will be tangled to

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

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

In the TEX (documentation) part of a section, '@|' 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 one wants.) For an example, see Section 5.12.4 [AT|_], page 56.

In a code part, '@|' inserts an optional line break in an expression—e.g.,

'f(a,b,@|c+d,...)'.

This helps TEX to break the line at an appropriate place. If the line does not need to be broken, the command does nothing. [Compare '@|' with '@\' (see Section 5.12.3 [ATbs], page 56) and '@/' (see Section 5.12.2 [AT/], page 55), which always break the line.]

5.3 Beginning of section

Sections are begun by either '@*' or '@␣'.

5.3.1 '@ ': Begin minor section

'@␣' begins a new minor (unstarred or unnamed) section that is not entered into the Table of Contents. For example,

```
@ This is an example of a minor (unnamed) 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 a period appears in the name itself, it must be protected by braces.)

The section name is also used as a running head on the output pages. To deal with the possibility that the full name may be too long, the section name may be preceded by an optional argument enclosed in brackets. If it is present, the optional argument is used as the running head. (If a period appears as part of the optional argument, it must be protected by braces.)

If '@*' is followed by a digit n, it begins a new major (sub)section of level n. This is also entered into the Table of Contents. Thus, the complete syntax to begin a major section is

@*n [Short name]Full name.

For example,

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

@a
main()
{}
```

```
@*1 [Input routines\dots]A very long section name that essentially
means ``input routines.''  Now follow some subroutines.

@a
get_input()
{}
```

For LaTeX, the highest permissible major level is 2 (a subsubsection).

Section names can contain reasonably arbitrary TeX text, including font-changing commands and other macros. However, it is necessary to understand that *fragile* commands (in the sense of LaTeX) may not work because the section name is used in various contexts (e.g., as a page header). If a macro in a section name doesn't work properly, try preceding it with '`\protect`'.

FWEAVE converts '`@*`' commands to section numbers. For a discussion of section numbering, see Section 10.1.6 [Numbering], page 98.

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 FWEB macro definitions begin with '`@#`', not '`@<`'.)

5.4.2 '`@>`': End module name

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

Technically, '`@>`' is not a command; rather, it is a delimiter that terminates '`@<`'. An unmatched '`@>`' is simply ignored (after a warning message is issued).

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' or `@A'.
@A
main()
{}
```

For more discussion of the distinction between '`@A`' and '`@a`', see Section 5.4.4 [ATa], page 40.

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@['

Originally, FWEB did not contain the '@A' command, so when the functionality of automatically marking the next unreserved identifier (see Section 5.7 [AT[]], page 51) was added, it was natural to add it to '@a'. A reasonable style of coding is to always use '@a' if you don't know any better; if you sometime run into trouble, you can then change selected '@a's to '@A's. For example, it is appropriate to use '@a' if one codes one function per section. E.g.,

```
@c
@
@a
int
subrtn()
{}
```

Here the '@a' marks 'subrtn' as defined in this section; if that identifier is used elsewhere, it will be subscripted with the section number. (To turn this feature off, use '-f'; see Section 4.2.16 [-f], page 20.) However, if a section contains an arbitrary code fragment, the code part should probably begin with '@A'. E.g.,

```
@c
@
@A
x = y;
```

If one had used '@a' here, the x would have been marked as defined here, which is not what one wants.

5.5 Control codes b–z

5.5.1 '@B': 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 the discussion of '@b' in Section 5.5.2 [ATb], page 41.

5.5.2 '@b': Insert a breakpoint command

(Discussion to be finished. Useful only for very intimate debugging of FWEB codes. In these days of safe sex, such intimacy may not be desirable.)

See also Section 5.5 [ATB_], page 41.

5.5.3 '@c': Set language to C

The command '@c' is a shorthand for '@Lc'. For a discussion of language commands in limbo, see Section 5.5.13 [ATL_], page 45.

See Chapter 8 [Languages], page 83 and Section 8.2.1 [C], page 84.

5.5.4 '@c++': Set language to C++

The command '@c++' is a shorthand for '@Lc++'. For a discussion of language commands in limbo, see Section 5.5.13 [ATL_], page 45.

See Chapter 8 [Languages], page 83 and Section 8.2.2 [Cpp], page 85.

5.5.5 '@D': Define outer macro

This command begins the definition part.

'@D' defines an outer macro. For more discussion, see Section 7.1 [Outer macros], page 62. For example, in C

```
@D A 1
```

will be tangled to the beginning of the output file as '#define A 1'.

5.5.6 '@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 identifier as defined in the present section. It is equivalent to

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

(see Section 5.7 [AT[], page 51).

The distinction between '@d' and '@D' is analagous to the distinction between '@a' and '@A'. See Section 5.4.4 [ATa], page 40.

5.5.7 '@E': Treat next identifier as ordinary expression (FWEAVE)

For formatting purposes, treat the next identifier as an ordinary expression.

This command is useful in pretty-printing certain kinds of macro constructions. Further discussion is given in ⟨undefined⟩ [Macros and formatting], page ⟨undefined⟩.

5.5.8 '@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++).

Traditionally, C programmers needed to use this command to format identifiers that were defined in `#include` files. This annoying redundancy has now been eliminated by means of the '-H' command, which tells FWEAVE to scan `#include` files automatically. See Section 4.2.17 [-H_], page 20.

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:

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

These are useful for dealing with certain macro constructions. For example,

```
@f PLUS $_BINOP_
@m PLUS +
@m ADD(x, y) ((x) PLUS (y))
```

Without the format command, the 'ADD' macro will pretty-print without spaces before and after the 'PLUS'.

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
```

Difficulties may ensue if one try to change the category code of '@' in this way; a fully operational WEB for TEX is quite difficult and has been neither accomplished nor attempted.

5.5.9 '@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 12.1 [Environment variables], page 107 and Section 4.2.19 [-I_], page 21.

In principle, the included file may contain any fragment of source text. However, it is best to make it a complete section (begun by '@*' or '@␣') if at all possible.

Unfortunately, the '@i' command cannot be commented out or conditionally included by use of an FWEB preprocessor command. That is because '@i' is processed very early in the parsing process. (Consider: '@i' could include TEX text, but the preprocessor is only active in the definition and code parts.)

Include commands may be nested to a depth set by the option '-yid'. See Section 12.2.2.9 [-yid], page 109.

In the woven output, if a section comes from an include file, the name of the include file is printed in square brackets as the first text of the TEX part. To inhibit printing of that name, say

```
\def\WIF#1{}
```

in the limbo section. To change the way that name is formatted, redefine the macro '\WIFfmt', whose single argument is the name of the include file. (It is not called when there is no current include file.) The default definition is

```
\def\WIFfmt#1{[{\tt#1}]}
```

5.5.10 '@I': Include file (conditional)

This command 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 4.2.20 [-i], page 21 and Section 4.2.21 [-i!], page 22.

5.5.11 '@K': Extract global RCS-like keyword

The construction '@K *Keyword* @>' accesses the value of a global RCS-like keyword. (For more discussion of such keywords, see Section 5.5.31 [ATz], page 50.) The command is treated differently by FTANGLE and FWEAVE depending on its location in the source file.

FWEAVE will expand the construction in the limbo section and TEX parts only. The value is not surrounded by quotes. For example,

```
@z
$Id:  test $
@x

@c

\def\ID{Id = \.{"@K Id @>"}}

@ \ID.  This is a @K Id @>.
```

will expand into

```
@c

@ \ID.  This is a test.
```

and when LaTEX is run the macro `\ID` will expand to '`Id = \.{"Test"}`'. The quotes are not necessary in the macro definition; they are included only to emphasize that in this (limbo) context the '@K' construction can effectively be put inside a string. This is possible because the routine that copies the limbo section simply copies characters; it does not tokenize anything.

FWEAVE does not expand '@K' constructions in the definition or code parts; it merely gives them a symbolic representation.

FTANGLE, on the other hand, expands '@K' constructions in the definition or code parts (during input). The values are surrounded by quotes. (As usual, FTANGLE ignores material in the limbo section and TEX parts.)

For FTANGLE, the built-in function '$KEYWORD' (see Section 7.2.3.30 [$KEYWORD], page 72) behaves essentially as does '@K', except that it is expanded during output, not input. FWEAVE does not expand '$KEYWORD'.

The command '@k' behaves as does '@K' except that it accesses local keywords, not global ones. See Section 5.5.12 [ATk], page 44.

5.5.12 '@k': Access local RCS-like keyword

The construction '`@k keyword`' behaves as '@K' does (see Section 5.5.11 [ATK_], page 44), except it accesses local keywords (defined at the top of include files).

5.5.13 '@L': Set language

'@Ll' sets the language to *l*, where *l* is one of '{c,c++,n,n9,r,r9,v,x}'. See Chapter 8 [Languages], page 83.

There are shorthand forms of this command for some languages; see '@c' (Section 5.5.3 [ATc], page 41), '@c++' (Section 5.5.4 [ATcpp], page 42), '@n' (Section 5.5.18 [ATn], page 47), '@n9' (Section 5.5.19 [ATn9], page 47), '@r' (Section 5.5.24 [ATr], page 49), and '@r9' (Section 5.5.25 [ATr9], page 49).

Generally, the global language should be set in the limbo section by means of '@L', '@c', etc. rather on the command line by options such as '-L' or '-c'.

5.5.14 '@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 '\n' (newline) 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. Sometimes, however, the '@l' command is useful for pedagogical purposes, as the limbo material can then be defined at the point where the logical discussion is made.

5.5.15 '@M': Define FWEB macro

This command begins the definition part.

For a detailed discussion of FWEB macros, see Chapter 7 [Macros], page 62.

5.5.16 '@m': Define FWEB macro, and mark

This command begins the definition part.

'@m' defines an FWEB macro, and also marks the next identifier as defined here. It is equivalent to

'@m' == '@M@['

(see Section 5.7 [AT[], page 51).

For a detailed discussion of FWEB macros, see Chapter 7 [Macros], page 62.

The distinction between '@m' and '@M' is analagous to the distinction between '@a' and '@A'. See Section 5.4.4 [ATa], page 40.

5.5.17 '@N': Turn on N mode

This command must appear before the code part. Generally, this means immediately before '@a'. *Do not use this command in limbo; use '@Lv' instead.*

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. FWEB 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 '@+'.
4. A module name must be within the scope of an '@N' the first time the name is seen, if it is ever to be within such scope. Thus, the following does not work properly:

   ```
   @ Consider the module @<Test@>.  (Not yet within scope of \.{@N}.)
   ```

```
@N
@a
x;
@<Test@>@;
y;
```

What happens is that the N mode is not restored after the code-part use of '@<Test@>'. This is a bug. There are very tricky design issues to be dealt with here.

5.5.18 '@n': Set language to FORTRAN–77

FORTRAN-77 is FWEB's default language, so this command is usually not strictly necessary. However, it is good practice to include it, so a user looking at the web file can tell immediately what language it is supposed to process.

For more discussion of languages, see Section 5.5.13 [ATL_], page 45 and Chapter 8 [Languages], page 83.

5.5.19 '@n9': Set language to FORTRAN–90

For more discussion of languages, see Section 5.5.13 [ATL_], page 45 and Chapter 8 [Languages], page 83.

For hints about FWEB programming in FORTRAN, see Section 8.2.3 [Fortran], page 85.

5.5.20 '@O': Open output file (global scope)

A statement of the form

`@O` *new_output_file_name*

changes the name of 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.

For an example of using the '@O' command to produce both C header files ('.h') and source files ('.c'), see the discussion in Section 7.1 [Outer macros], page 62.

To change the name of the output file locally (for just the present section), see Section 5.5.21 [ATo], page 47.

5.5.21 '@o': Open output file (local scope)

This behaves like '@O', except that the new file name is in effect only for the current section. A subsequent '@o' issued in a different section but for the same file name accretes material to the file.

An annoying problem arises in C programming when '@o's are used to create multiple source files that are subsequently compiled under the control of a `Makefile`. Remember that by default line-number information is written into the C files. This means that a change in the web file code for one source file can affect all of the others, because the line numbering in the web file changes. Therefore, a trivial change to the code for just one source file can cause all of the others to be recompiled.

As long as one desires debugging information relative to the original `web` file, there is really no solution to this problem; one needs the proper line information in each file in order to work with the debugger, so if line numbers change the sources must be recompiled. One can, of course, turn off the line numbering with the command-line option '`-#`' (see Section 4.2.78 [-#], page 36), but then debugger statements will refer to the tangled C code, which is undesirable. A better partial solution is to use '`@q`' (see Section 5.5.22 [ATq], page 48) to turn off the line numbering for output code that is currently stable. In the following example, the code for each file is put into a module, then the modules are output in the unnamed section; it is assumed that the programmer is currently making changes to the code for '`file2.c`':

```
@
@a
@q0
@o file1.c
        @<File 1@>@;
@q1
@o file2.c
        @<File 2@>@;
@q0
@o file3.c
        @<File 3@>@;
```

For very large projects, another solution is to maintain multiple `web` source files. To avoid losing the substantial benefits of the automatic index, refer to the discussion in Section 11.3 [Merging indexes], page 105 to learn how to create a master index that contains information about several `web` files.

5.5.22 '@q': Turn off module and line info locally

The command-line option '`-#`' (see Section 4.2.78 [-#], page 36) turns off comments about module and line numbers globally, for the entire code. However, in some cases one wants to turn that off in just a small block of code. One important example arises in FORTRAN. Consider

```
@
@a
      x = @<Some action@>

@
@<Some action@>=
y + z
```

This example will tangle to something like

```
      x =
C* 1: *
*line 20 "test.web"
      y + z
C* :1 *
*line 5 "test.web"
```

Unfortunately, the information comments have created invalid code that will not compile.

The '@q' command solves this problem by turning off or on the information comments locally. '@q0' turns them off; '@q1' turns them on. Thus, if one rewrites the above example as

```
@
@a
@q0
        x = @<Some action@>
@q1
```

it will tangle to

```
        x = y + z
```

as one desires.

For another use of the '@q' command, see Section 5.5.21 [ATo], page 47.

5.5.23 '@R': Treat next identifier as integer-like reserved

word (FWEAVE)

For formatting purposes, treat the next identifier as an integer-like reserved word.

This command is useful in pretty-printing certain kinds of macro constructions. Further discussion is given in ⟨undefined⟩ [Macros and formatting], page ⟨undefined⟩.

5.5.24 '@r': Set language to RATFOR–77

See Section 5.5.13 [ATL_], page 45 and Chapter 8 [Languages], page 83.

5.5.25 '@r9': Set language to RATFOR–90

See Section 5.5.13 [ATL_], page 45 and Chapter 8 [Languages], page 83.

5.5.26 '@u': Undefine outer macro

This command begins the definition part.

'@u' is the inverse of '@d'. For example, in C the command '@u A' tangles to '#undef A'.

5.5.27 '@v': Overload operator

This command begins the definition part.

'@v' is used to change the woven appearance of an operator. If one defines a new operator, for example by a statement such as

```
interface operator(.BETA.)
```

in FORTRAN-90, one should also use an '@v' in the definition part—for example,

```
@v .BETA. "\\beta" +
```

For a detailed discussion of overloading (the output appearance of) operators, see Section 10.2.3 [Overloading], page 101.

5.5.28 '@W': Overload identifier

This command begins the definition part.

For a detailed discussion of overloading (the output appearance of) identifiers, see Section 10.2.3 [Overloading], page 101.

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

In a change file, this command begins material to be changes; see Section 3.3 [Change files], page 13.

In `web` source files, this command has a different use; see the discussion of the '`@z`' command (see Section 5.5.31 [ATz], page 50).

5.5.30 '@y': Begin change material

The '`@y`' command is permitted only in change files. See Section 3.3 [Change files], page 13.

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

FWEB files may begin with the construction

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

where the '`@z`' occupies the very first two characters of the file, and 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, with one exception.

The exception is that an RCS-like line (RCS stands for "revision-control system") with syntax

```
$Keyword: Text of Keyword $
```

(at least one blank after the colon, and at least one before the last dollar sign; UNIX users, see '`man ident`') is parsed, and the text of the `Keyword` is made available to the control codes '`@K`' (see Section 5.5.11 [ATK_], page 44) and '`@k`' (see Section 5.5.12 [ATk], page 44) as well as to FTANGLE's built-in function `$KEYWORD` (see Section 7.2.3.30 [$KEYWORD], page 72.

A distinction is made between keywords that are found in the ignorable commentary at the beginning of the master web file, which are called *global keywords*, and ones that are found at the beginning of files included via '`@i`', which are called *local keywords*.

The commands that access RCS-like keywords function as follows:

- '`$KEYWORD(Keyword)`' accesses a global keyword. It is a built-in function that is expanded by FTANGLE (during output) into the quoted character string `"Text of Keyword"`.
- '`@K`' and '`@k`' are expanded during input. '`@K`' accesses a global keyword, whereas '`@k`' accesses a local keyword.

- In the limbo section or a TeX part, FWEAVE will expand '`@K Keyword @>`' into `Text of Keyword` (without the surrounding quotes), and similarly for '`@k`'. (The intention is that the expanded text can be used as bodies of TeX macros.) FWEAVE will also print the values of global keywords at the end of its output, whether or not they are referenced by '`@K`'.
- Elsewhere FWEAVE will just print the keyword name itself, surrounded by double angle brackets. If the keyword was local ('`@k`'), the brackets will carry the subscript 0.
- FTANGLE treats the global command '`@K Keyword @>`' essentially like it does '`$Keyword`', except that the construction is expanded on input rather than output.
- FTANGLE expands the command '`@k keyword @>`' on input, generating a quoted string containing the value of the local keyword.

The command '`@z`' is also used in change files to end a change. See Section 3.3 [Change files], page 13.

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 one's machine is ASCII, the conversion will not be done unless the '`-A`' command-line option is used. See Section 4.2.5 [-A_], page 17.

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 12.3.8.1 [ASCII_fcn], page 118.

If the native character set of one's machine is ASCII, the conversion will not be done unless the '`-A`' command-line option is used. See Section 4.2.5 [-A_], page 17.

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@[', '@d' is equivalent to '@D@[', and '@m' is equivalent to '@M@['.

If the appropriate style-file parameter mark_defined.??? is 1, this command causes any appearance of the identifier to be subscripted with a section number. For more information, see Section 12.3.4 [Subscript params], page 114.

The utility of this command can be seen from the characteristic construction

```
@ This is section 5.
@a @% Issues an implicit @[, which marks |test| as defined in section 5.
        subroutine test
        ...
        end

@ This is section 6.
@a
        program main
        call test // This will print as $|test|_5$.
        end
```

The '@[' command should be distinguished from '@_' (see Section 5.10 [AT_], page 54). The latter causes the index entry for the identifier to be underlined; the former possibly causes the identifier to be subscripted by a section number. One may wish to turn off the subscripts because they become too cluttered; however, the underlined index entries remain useful and unobtrusive.

5.8 Comments

FWEB supports a variety of commenting styles borrowed from C, C++, and TEX. For more discussion, see Chapter 6 [Comments], page 60.

5.8.1 '@/*': Begin long verbatim comment

The following comment is copied to the tangled output. (By default, comments are not copied.) If you desire all comments to be so copied, use '-v'. See Section 4.2.64 [-v], page 32.

5.8.2 '@//': Begin short verbatim comment

See the discussion of '@/*' in Section 5.8.2 [AT//], page 52.

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 5.5.17 [ATN_], page 46.

Line-numbering problems can arise when these commands are used. For a discussion, see Section 4.2.59.6 [-T#], page 31.

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 12.3.8 [Miscellaneous params], page 118.

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 12.3.8 [Miscellaneous params], page 118).

5.8.6 '@)': End meta-comment

See the discussion of '@(', Section 5.8.5 [ATlp], page 53.

5.9 Special left brace

The command '@{' is useful in C/C++ programming to beautify some of the pretty-printing. It translates into a left brace, but also suppresses the automatic insertion of newlines into the subsequent function body or block. This is desirable for very short functions, such as simple constructors in C++. For example,

```
class C
{
private:
        int i;

public:
        C(int i0) @{i = i0;}
}
```

Here the function will be typeset as

```
C(int i0)
  { i = i0; }
```

rather than the default

```
C(int i0)
  {
  i = i0;
  }
```

5.10 Index entries

Although most information for the Index is gathered automatically, in some situations it must be done by hand.

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. (By convention, this means 'defined' or 'declared'.)

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, since FWEAVE understands enough of the syntax to know that variables are being defined. Macro definitions (begun by '@D' or '@M') will also be underlined automatically.

5.10.2 '@-': Delete index entry

This command applies to the next identifier that appears after the '@-'; it prevents an index entry associated with that identifier from being made. 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 (user-defined format)

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

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

where argument '#1' is the text between '@9' and '@>'. For example, to print that text in a sans serif font, say

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

(Note the extra level of braces to prevent the font command from propagating.)

5.11 Control text

Control text is material terminated by '`@>`'; it must be all on one line and must not contain any '`@`'s.

5.11.1 '`@t`': Put control text into a TEX \hbox (FWEAVE)

When FWEAVE sees the command '`@t`*control text*`@>`', 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 '`@=`*control text*`@>`' sends the control text to the output exactly as input. 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 one is 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. '`@,`' inserts a thin space, analogous to TEX's `\,`.

An example where explicit spacing would be necessary is as follows:

```
@c
@
@m OP +
@m A(x,y) x @, OP @, y

@a
z = A(a, b);
```

Without the '`@,`''s, the body of the `A` macro will weave as the unappealing '`xOPy`'. This occurs because although `OP` is defined to be a binary operator, FWEAVE thinks of it as just a mere expression, and one of its fundamental production rules is to concatenate expressions with no intervening expressions.

This demonstrates that situations arise in which one needs to override FWEAVE's default processing. But for the above example, there is actually a better solution. Instead of using the '`@,`''s, include the format command '`@f OP $_BINOP_`'. See Section 5.5.8 [ATf], page 42.

5.12.2 '`@/`': Force a line break, preserving indentation.

This command is used to override FWEAVE's natural inclinations. For example, if one wants each piece of a declaration to appear on a separate line, one can say

```
int@/
  i,@/
  j,@/
```

```
k;
```

This command preserves the natural indentation that would have happened if FWEAVE or LaTEX had broken a long line spontaneously. Thus, the declared variables are indented in the above example. To remove that indent, use '`@\`' instead. See Section 5.12.3 [ATbs], page 56.

Try to use the line-break commands sparingly—i.e., let FWEAVE do the work. Often, if lines run together in an unexpected or unreadable way, it's because FWEAVE wasn't able to parse the relevant block of code, perhaps because it didn't understand that some variable in an include file has a special meaning. In such cases, trying to fix things with '`@/`' is the wrong solution. Either use '`@f`' (see Section 5.5.8 [ATf], page 42) or '`-H`' (see Section 4.2.17 [-H_], page 20).

Distinguish the '`@/`' command from '`@|`' (see Section 5.2.2 [AT|], page 39), which inserts an optional breakpoint into an expression.

5.12.3 '`@\`': Force a line break, then indent

The '`@\`' command behaves like '`@/`' (see Section 5.12.2 [AT/], page 55), except that it backspaces one notch after the line break. This usually has the effect of undoing the natural indentation that would have been inserted had a long line been spontaneously broken. One common case where the '`@\`' command might be used would be to put the return type of a C function on a separate line:

```
int @\
main()
{}
```

It would be nice to have FWEAVE do that automatically. Unfortunately, the syntax of a function isn't recognized until the opening braces are sensed; by that time, the declaration part of the statement has already been processed. This is one example of the fact that the FWEB processors are much less intelligent and sophisticated than language compilers. A clever (and simple) idea for getting around this kind of problem is lacking at this point.

5.12.4 '`@|`': Literal vertical bar, or optional line break

In the TEX (documentation) part of a section, '`@|`' 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 a code part, '`@|`' inserts an optional line break in an expression.

5.12.5 '`@#`': Blank line

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

if '`@#`' is immediately followed by a letter (e.g., '`@#if`'), it is assumed that a preprocessor command is beginning. See Section 7.3 [Preprocessing], page 80.

5.12.6 '@~': Cancel line break

'@~' is analogous to TeX's '~' (tie); it prevents a line break, which FWEAVE usually inserts after each complete statement it recognizes. For example,

```
printf("Working..."); @~ fflush(stdout);
x = y; @~ break;
```

5.12.7 '@&': 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 an FWEB macro that should itself be expanded.

5.13 Pseudo (invisible) operators

Pseudo- or invisible operators are ignored by FTANGLE and not printed by FWEAVE; however, they retain grammatical significance that helps out FWEAVE in its attempts to understand the syntax.

5.13.1 '@e': Pseudo-expression

'@e' is an invisible expression ('pseudo-expression') (see Section 10.2.1 [Pseudo-operators], page 101). It is sometimes useful in situations where FWEAVE's pretty-printing has broken down because it didn't properly understand the language syntax. If, for example, FWEAVE failed to properly parse the C statement

```
p = (int (*))q;
```

one might get things to work properly by saying

```
p = (int (*@e))q;
```

In this particular case, one is patching up a deficiency (all right, a bug) in FWEAVE's "production rules." (This particular bug may no longer exist.) However, there are other situations in which the use of '@e' might be necessary. Consider, for example, the C macro definition

```
#define A(x) = x
```

Here the replacement text of the macro is '= x', which by itself is not a valid construction in C. When the '-1' or '-2' options are used, FWEAVE will report an "irreducible scrap sequence" in this situation (although it may typeset it correctly anyway). To eliminate the warning message, say instead

```
#define A(x) @e = x
```

Now the fragment '@e = x' is interpreted as a valid expression.

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. Carefully compare the uses of '@;' and ';' in the following example:

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

@ This compound statement ends with a brace, but is used as an
expression above.
@<Com...@>=

{
x;
y;
}

@ This fragment does not end with a semicolon, so one must be
supplied above.

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

Here is a case for which the pseudo-semicolon is not necessary. Consider

```
@c
@ The code fragment |x = y| ...
```

If the '-1' is turned on, one might think that FWEAVE would report an "irreducible scrap sequence" because 'x = y' is an expression but not a complete statement. (Turning on '-2' demonstrates this.) However, it is not necessary to say '|x = y@;|' because the warning message is not issued if the parsing reduces to just one unresolved scrap.

On the other hand, '|goto done|' does not reduce to just one unresolved scrap, so say '|goto done@;|' in cases such as this. See Section 10.2.1 [Pseudo-operators], page 101.

In some situations, pseudo-semicolons are inserted automatically. An important case is free-format FORTRAN-90. There the language syntax says that newlines terminate statements (except when there's a trailing ampersand). However, newlines are thrown away before tokenized text is seen by FWEAVE's parser (and in any event would just be interpreted as white space). Therefore, by default newlines that terminate statements are replaced by pseudo-semicolons, so the parsing proceeds correctly.

In the FORTRAN-90 case, one could also insert pseudo-semicolons or actual semicolons by hand, and some users prefer that. The possibilities are controlled by the options '-n@;' (see Section 4.2.32 [-nAT;], page 24) and '-n;' (see Section 4.2.33 [-n;], page 24).

5.13.3 '@:': Pseudo-colon

'@:' is an invisible colon (see Section 10.2.1 [Pseudo-operators], page 101). It can be helpful in formatting certain C constructions correctly. For example, if one has a named module defined as

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

then one can use it as a case construction followed by the usual colon, as in

```
switch(c)
        {
   @<Cases@>:
        stuff;
        break;
        }
```

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 `\mainfont`, a ten-point Roman font.

6.1 Invisible comments

`@z...@x` If a source or include file begins with '`@z`' (in the very first two characters of the file), then all material is skipped until and including a line beginning in column 1 with '`@x`' [except that lines of the form '`$Keyword: text of keyword $`' are processed; see Section 7.2.3.30 [$KEYWORD], page 72, Section 5.5.11 [ATK_], page 44 (source files), or Section 5.5.12 [ATk], page 44 (include files)].

`@%` 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 (it may extend over several lines).

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

'`@(...@)`' is a meta-comment. Meta-comments are a localized form of the N mode (see Chapter 8 [Languages], page 83). 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 12.3.8 [Miscellaneous params], page 118.

```
@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-person'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; they are only marginally supported. Use ordinary long comments instead. Inside of them, use the various powerful features of TEX or LaTEX (such as `\halign` or `\begin{verbatim} ... \end{verbatim}`) to format your comment appropriately.

6.3 Temporary comments

During development, one frequently desires to temporarily comment out a section of code. C programmers sometimes try to do this by enclosing the code in `/*...*/`, but this is *not* good style for several reasons. First, it is impossible if the code itself includes comments, since comments do not nest in C. Second, FWEAVE will treat the commented code as TEX rather than C code and will (at best) format it very poorly. In fact, LaTEX will frequently complain, because the commented code might contain characters such as underscores that TEX expects to be in math mode. (Those are dealt with automatically when FWEAVE is in code mode.) The trivial example '`/* a_b; */`' is sufficient to illustrate this point.

The proper way of commenting out sections of code is to use preprocessor constructions: `#if 0...#endif` in C, or more generally `@#if 0...@#endif` (usable in all languages). (The FWEB preprocessor is described in Section 7.3 [Preprocessing], page 80.) With this method, there is no trouble with nested comments, and FWEAVE will continue to format the code as code, so the documentation will make sense.

For FORTRAN programmers converting an existing code to FWEB, the '`-nC`' option (see Section 4.2.36 [-nC], page 25) may be helpful.

7 MACROS and PREPROCESSING

FWEB offers a built-in preprocessor facility, especially useful for FORTRAN programmers. It is closely patterned after the C/C++ preprocessor, but with some extensions such as variable numbers of arguments. In addition, there are some built-in functions that provide functionality that cannot be emulated by user-defined macros.

When working with a language such as C that has its own preprocessor, the question arises when to use that and when to use FWEB's facilities. The answer generally comes with experience. Remember that FWEB's macros have been expanded by the time the tangled output file is produced, whereas language-specific preprocessor commands are just passed through to that file.

If you're a FORTRAN programmer, *strongly* consider the use of FWEB's macro facilities; they will simplify your present and future life by creating more legible codes and reducing programming errors by eliminating redundant pieces of code. C/C++ programmers may also appreciate the preprocessor extensions.

In addition to conventional macro processing, FWEB also offers the convenience of certain built-in functions that behave in many ways like macros. As a trivial example, the value of π is available through the built-in function '`$PI`'. Built-in functions are described in Section 7.2.3 [Built-in functions], page 66. They can be useful to programmers in all languages.

FWEB recognizes two kinds of macros: *outer macros*, and *WEB macros* (*inner macros*). Control codes associated with either of these kinds normally begin the definition part. However, FWEB macros are sometimes allowed in the code part as well; see Section 7.2 [FWEB macros], page 63.

Macros are expanded by FTANGLE only; FWEAVE merely prints them as they occur in the source file.

7.1 Outer macros

Outer macros provide a shorthand way of invoking macro definitions in the source language; they are not expanded by FWEB. Outer macros are defined by '`@d`' (see Section 5.5.6 [ATd], page 42) or '`@D`' (see Section 5.5.5 [ATD_], page 42). They may be placed in any definition part. FTANGLE collects them 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 quick 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 12.3.8 [Miscellaneous params], page 118.

Outer macros can be undefined by '@u'. The translation is controlled by the style-file parameter 'outer_undef'. See Section 12.3.8 [Miscellaneous params], page 118.

The default behavior, in which the outer macro definitions are just copied to the top of the output file, is fine for simple applications. However, often C programmers prefer to maintain their macro definitions in a header file such as 'test.h'. One way of accomplishing this is to redirect FTANGLE's output from the command line, as in 'ftangle test -=test.h', then use an '@O' command immediately after the first '@a' in the web file to open up 'test.c'. A more complicated variant of this allows additional information to be placed into the header file, as in the following example:

```
@c
@* INTRO.
We assume command-line redirection into \.{test.h} (`\.{-=test.h}').

@d A 1 // This will go into \.{test.h}.

@a
@<Header material@>@; // Also goes into \.{test.h}.
@O test.c // Remaining unnamed sections go into \.{test.c}.

@ Header material may be defined as needed throughout the code, but
with this design it will all go into \.{test.h}.

@<Header material@>=

@<Includes@>@;
@<Typedefs@>@;
@<Global variables@>@;
```

7.2 FWEB macros

FWEB macros (sometimes called *inner macros*) are defined by '@m' (see Section 5.5.16 [ATm], page 45) or '@M' (see Section 5.5.15 [ATM_], page 45). These should normally be placed in the definition part, as in

```
@n
@ Documentation...

@m CUBE(x) (x)**3

@a
        z3 = CUBE(x) + CUBE(y)
```

(the appearance of an '`@m`' in the documentation part begins the definition part). They are collected during FTANGLE's phase 1 and effectively 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 FWEB 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.

The use of deferred macros is highly discouraged, for the following reason. FWEB macros are often used in conjunction with the FWEB 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 FWEB macros

- Fundamentally, FWEB macros follow the syntax for ANSI C. There are also a few extensions, notably the possibility of variable (optional) arguments (see Section 7.2.1.1 [Variable arguments], page 64) and some additional preprocessing tokens (see Section 7.2.2 [Tokens], page 65).
- Adjacent strings in macro text are automatically concatenated.

7.2.1.1 FWEB macros with variable arguments

An important extension to the ANSI-C syntax is to allow macros with variable (optional) arguments. FWEB macros with a variable number of arguments are indicated by an ellipsis, as in

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

The tokens '`#0`' (number of variable arguments), '`#`*n*' (value of the *n*th optional argument), and '`#.`' (comma-delimited list of the optional arguments) are useful in this context.

7.2.1.2 Recursion

ANSI C does not permit recursive macros (for good reason). Thus, in the example

```
@m recurse recurse
```

the identifier `recurse` simply expands as '`recurse`', not as an infinite loop. However, in FWEB recursion may be useful in conjunction with some of the built-in functions (see Section 7.2.3 [Built-in functions], page 66). To permit a macro to be recursive, say '`@m*`'.

No formal support is provided for recursive macros! If they don't work, or suddenly stop working in a new release, *you're on your own*!

7.2.1.3 Protecting macros against redefinition

Normally an FWEB macro can be redefined at will. The example

```
@m PI 3.14159
@m PI (-3)
```

is permissible, but probably not a good idea. If you want to ensure that a crucial macro definition is never redefined inadvertently, say '`@m!`', as in

```
@m! PI 3.14159
```

That is called *protecting* the macro.

FWEB's built-in functions and macros (beginning with '`$`') are protected by default; see Section 7.2.3.2 [Protection], page 67. To override that protection, use the command-line options '`-Tb`' (Section 4.2.59.2 [-Tb], page 31; for built-in functions) or '`-Tm`' (Section 4.2.59.3 [-Tm], page 31; for macros).

7.2.2 Special tokens

The following special tokens may be used in the text of FWEB 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).

For example,

```
@m FORTRAN(type, name) type _##name()
@m TRACE(where) puts("At " #where)
@a
FORTRAN(int, fcalc); // Expands to 'int␣_fcalc();'
TRACE(predictor); // Expands to 'puts("At␣"␣"predictor");'
```

7.2.2.2 Extensions to ANSI C macro syntax

The most frequently used extensions are the following ones associated with variable arguments: '`#0`', '`#`*n*', and '`#.`'. FORTRAN-77 users should also employ '`#:0`' to allow symbolic rather than numeric statement labels. Try not to use the other extensions; they are experimental, complicated, and unlikely to work in all situations.

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

`#*`*param* Like '`#`*parameter*', but pass a quoted string through unchanged.

`#!`*param* Don't expand argument.

`#'`*param* Convert parameter to a single-quoted string (no expansion).

`#"`*param* 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 the 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.

A few examples of the more important of these tokens are as follows:

```
@c
@m FPRINTF(fmt,...) fprintf(fp,fmt,#.)
        // Use the whole list of variable args.
@m B(...) printf("There were %i arguments\n", #0)
        // Use the number of var args.

@n
@
@m DONE #:0 // Symbolic statement label in Fortran.
@a
        goto DONE
        ...
DONE:
        call endup
```

7.2.3 Built-in functions

Built-in functions behave in most ways like macros. In some cases they actually are macros, but other times they implement functions that a user could not define. They all begin with a dollar sign and are in upper case.

In using these built-ins, confusion may arise regarding the order of expansion of various arguments. When they are implemented as macros, they are subject to the same ANSI-C preprocessor rules as other FWEB macros, which is that all arguments are fully expanded before generating the replacement text of the macro. When they are directly implemented as a primitive function, however, that rule may not apply. For example, `$IF` expands only its first argument during its first pass of processing; depending on the results of that expansion, it then expands *either* its second or third argument, but not both.

The built-in function `$DUMPDEF` can be used to understand and debug the action of the built-in functions. See Section 7.2.3.14 [$DUMPDEF], page 70.

In the original FWEB design, built-in functions began with an underscore. This usage conflicts with the conventions for reserved words in ANSI C, and has been eliminated. *All* FWEB *built-ins now begin with a dollar sign.*

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

7.2.3.1 Strings and quotes

Several of the built-in functions expect or return a string argument. Examples include `$STRING` (see Section 7.2.3.58 [$STRING], page 77), `$UNQUOTE` (see Section 7.2.3.64 [$UNQUOTE], page 78), and `$UNSTRING` (see Section 7.2.3.65 [$UNSTRING], page 78). In understanding the operation of those functions, it is important to understand just what a string means in the FWEB context. As usual, it is a vector of characters. However, *those need not be delimited by quotes*, although they may be. Internally, a string is represented by the construction *sqc...cqs*, where *s* is a special string delimiter never seen by the user, *q* is an optional quote character (either single or double quote depending on the language), and *c* is an ordinary character. Whether or not the quotes are present, the string delimiters inhibit macro expansion.

The difference between `$UNQUOTE` and `$UNSTRING` can now be stated as follows. Given a quoted string such as `"abc"` (in C),

- '`$UNQUOTE`' removes the quote characters *q*, leaving *sabcs* (still a string).
- '`$UNSTRING`' removes both the quote characters *q* and the string delimiters *s*, leaving *abc* (a collection of three characters). This collection is *not* tokenized; it does *not* represent the single identifier name `abc` (and therefore is not very useful). `$UNSTRING` is primarily used internally.

The built-ins `$P` (see Section 7.2.3.46 [$P], page 76) and `$PP` (see Section 7.2.3.49 [$PP], page 76), which both generate the preprocessor character '`#`', provide a good illustration of the differences between `$UNQUOTE` and `$UNSTRING`. Consider FORTRAN as an example. Essentially, `$P` is defined as '`$UNQUOTE('#')`', which is internally *s*`#`*s*. When this single-character string is sent to the output, it is treated like any other expression and therefore would appear in column 7 or greater even if the construction appeared at the very beginning of the line. On the other hand, `$PP` is (essentially) defined as '`$UNSTRING('#')`', which is internally the single character `#`. Because this character is not a string, the FORTRAN output driver treats it as a special control character, defined in this case to force the character into the first column.

7.2.3.2 Redefining built-in functions

By default, built-in functions are *protected*—that is, they may not be redefined by an `@m` command. (To do so cavalierly invites many kinds of weird disasters.) If it is absolutely necessary to redefine a built-in function, use the command-line option '`-Tb`' (see Section 4.2.59.2 [-Tb], page 31).

Many of FWEB's "built-in functions" are in fact ordinary macros that are implemented in terms of lower-level built-ins. An example is `$POW` (see Section 7.2.3.48 [$POW], page 76), which is constructed from the built-in function `$EVAL` (see Section 7.2.3.17 [$EVAL], page 70). By default, such macros are also protected against redefinition; to override, use the option '`-Tm`' (see Section 4.2.59.3 [-Tm], page 31).

7.2.3.3 `$A`: Convert to ASCII

'`$A(`*string*`)`' is the built-in equivalent of '`@'...'`' or '`@"..."`'. (See Section 5.6 [ATquote], page 51 and Section 5.6.2 [ATdquote], page 51.) Note the extra parentheses required by the built-in.

`$A` first expands its argument, in case it is a macro defined as a string.

7.2.3.4 `$ABS`: Absolute value

'`$ABS(`*expression*`)`' returns the absolute value of the macro expression. It is a macro implemented in terms of `$IF` and `$EVAL`.

7.2.3.5 `$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 FWEB macros required by the code are properly initialized. Because it is expanded during the output phase, it must appear in the *code part* (not in the definition part).

7.2.3.6 `$AUTHOR`: Value of RCS global keyword `Author`

Equivalent to '`$KEYWORD(Author)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.7 `$COMMENT`: Generate a comment

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

This function is sometimes useful in conjunction with the processing of FWEB macros, since ordinary comments are removed when macros are processed. For example, if one says

```
@c
@
@m M "abc" $COMMENT("Test")
@a
m = M
```

the tangled output will be '`m= "abc"/* Test */`'

7.2.3.8 `$DATE`: Today's date

'`$DATE`' generates a string consisting of the date in the form `"August 16, 2001"`. It is implemented as a macro that calls other macros and primitive functions.

7.2.3.9 `$DATE_TIME`: Value of RCS global keyword `Date`

Equivalent to '`$KEYWORD(Date)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.10 `$DAY`: The day

'`$DAY`' generates a string consisting of the day of the week, such as `"Monday"`. It is implemented as a macro that calls other macros and primitive functions.

7.2.3.11 `$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*'.

7.2.3.12 `$DEFINE`: Deferred macro definition

'`$DEFINE`' behaves like the FWEB macro command `@m`, but it is intended to appear in the *code* part, not the definition part (so it is processed during *output*, not input). Thus, the code fragment

```
a = A;
$DEFINE(A 1)@%
a = A;
```

tangles to

```
a= A;
a= 1;
```

(Notice how the '`@%`' command was used to kill an unwanted newline, analogous to the '`dnl`' macro in `m4`.)

In the above example, one could also say '`$DEFINE(A=1)`'. To define a macro with arguments, say something like '`$DEFINE(A(x)x*x)`'. Do *not* say '`$DEFINE(A(x)=x*x)`', as in this case the equals sign will be included in the macro expansion. One must use the equals sign as a means of preventing parentheses from being interpreted as an argument in examples like

```
$DEFINE(A=(x))
```

This expands to '`(x)`'.

A completely equivalent shorthand notation for `$DEFINE` is `$M`.

7.2.3.13 `$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;
```

In general, the macro name used as loop counter should *not* be explicitly defined as a macro prior to the `$DO`. If it is not, it will remain undefined 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 if only a single statement is to expanded.

`$DO` is implemented in terms of a command `$UNROLL`. However, if one says something like '`$DUMPDEF($UNROLL(0,5,1))`', FWEB will respond that `$UNROLL` is not an FWEB macro. Rather, `$UNROLL` is processed like expandable commands in RATFOR such as `while`. This implies that it cannot be redefined as ordinary macros or built-in functions can be.

7.2.3.14 `$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.

One can use this built-in to debug one's own macros, or to find out the secrets of FWEB's built-ins. As an example, if one says

```
$DUMPDEF($EVAL(2^^4))@%
```

it responds with the two lines

```
$EVAL($0) = $$EVAL($0)
$EVAL(2**4) = 16
```

(The `$`*n* notation indicates the *n*-th argument of the macro.) If one replaces `$EVAL` with `$$EVAL` in the above `$DUMPDEF`, it will respond

```
$$EVAL($0) = <built-in>
$$EVAL(2**4) = 16
```

The purpose of code such as '`$EVAL($0) = $$EVAL($0)`' is to ensure that the argument of `$EVAL` is expanded if it contains macros; the primitive function `$$EVAL` does not do that expansion automatically.

Names indicated as '`<built-in>`' by `$DUMPDEF` may be redefined as ordinary macros, but this is in general a *very bad idea*; other parts of FWEB may mysteriously stop working.

7.2.3.15 `$E`: Base of the natural logarithms

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

7.2.3.16 `$ERROR`: Send error message to output

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

7.2.3.17 `$EVAL`: Evaluate a macro expression

'`$EVAL(`*expression*`)`' uses FWEB's macro-expression evaluator (see Section 7.3 [Preprocessing], page 80) 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.

7.2.3.18 `$EXP`: Exponential function

'`$EXP(`*x*`)`' returns e^x.

7.2.3.19 `$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.)

The argument to `$GETENV` need not be a string (double-quoted), but it may be if necessary to avoid the expansion of a macro.

7.2.3.20 `$HEADER`: Value of RCS global keyword `Header`

Equivalent to '`$KEYWORD(Header)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.21 `$HOME`: The user's home directory

'`$HOME`' is a convenience macro equivalent to '`$GETENV(HOME)`'.

7.2.3.22 `$ID`: Value of RCS global keyword `Id`

Equivalent to '`$KEYWORD(Id)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.23 `$IF`: Two-way conditional

`$IF` is a primitive function (not a macro) that is the code-part version of '`@#if`'. The syntax is

`$IF(`*expr*`, `*action-if-true*`, `*action-if-false*`)`

The *expr* is an FWEB macro expression that must reduce to 0 (false) or 1 (true). First that argument is expanded. If it is true, *action-if-true* is expanded; otherwise *action-if-false* is expanded.

There may be peculiarities with this and the other built-in `$IF` function having to do with the order of expansion when the actions contain macros whose arguments themselves are macros. Therefore, do not use them unless absolutely necessary.

Do not redefine `$IF` or any other built-in conditionals, as they are used internally to FWEB.

7.2.3.24 `$IFCASE`: n-way conditional

This primitive built-in behaves like TeX's '`\ifcase`' command. The syntax is

`$IFCASE(`*expr*`, `*case-0*`, `*case-1*`, ...,`*case-n-1*`, `*default*`)`

If *expr* reduces to an integer between 0 and *n-1*, inclusively, the appropriate case is selected; otherwise, the default case is selected.

As examples,

```
$IFCASE(2, zero, one, two, default) => 'two'
$IFCASE(2, zero, one, three) => 'three'
$IFCASE(2, zero, one) => 'one'
```

7.2.3.25 `$IFDEF`: Two-way conditional

This built-in primitive is the code-part version of '`@#ifdef`'. The syntax is

`$IFDEF(`*macro*`, `*action-if-defined*`,`*action-if-not-defined*`)`

7.2.3.26 `$IFNDEF`: Two-way conditional

This built-in primitive is the code-part version of '`@#ifndef`'. The syntax is '`$IFNDEF(`*macro*`, `*action-if-not-defined*`, `*action-if-defined*`)`'.

7.2.3.27 `$IFELSE`: Two-way conditional

The syntax of this built-in primitive is '`$IFELSE(`*expr1*`, `*expr2*`, `*action-if-equal*`, `*action-if-not-equal*`)`'. The expansions of *expr1* and *expr2* are compared on a byte-by-byte basis. If they are equal, the first action is taken, otherwise the second action is taken.

For example,

```
$M(S="abc")@%
$IFELSE("abc", S, yes, no)
```

evaluates to '`yes`'.

7.2.3.28 `$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*'.

7.2.3.29 `$INPUT_LINE`: Line number that begins current section

'`$INPUT_LINE`' is the number of the line in the `web` source file that *begins the current section* (not the source line in which the `$INPUT_LINE` command appears). Compare `$OUTPUT_LINE`, Section 7.2.3.45 [$OUTPUT_LINE], page 76.

7.2.3.30 `$KEYWORD`: Value of global RCS-like keyword

'`$KEYWORD`' provides a built-in function alternative to the use of '`@K`' in a code part. (see Section 5.5.11 [ATK_], page 44).

'`$KEYWORD(`*Keyword*`)`' extracts (as a character string) the text of an RCS-like keyword defined in the ignorable commentary between '`@z`' and '`@x`' at the beginning of the web source file (see Section 5.5.31 [ATz], page 50). (*RCS* stands for "revision-control system.") The general syntax is (UNIX users, see '`man ident`')

`$`*Keyword*`:` text of keyword `$`

For example,

```
@z
$Author: krommes $
@x
```

```
@c
@
@a
char author[] = $KEYWORD(Author);
```

This tangles to

```
char author[] = "krommes";
```

In this example, '`$Author`' is one of the standard RCS keywords. However, any keyword that fits the syntax '`$`*keyword*`:` *contents* `$`' can be accessed by '`$KEYWORD`'. (At least one blank is necessary before and after *contents*.) The argument of '`$KEYWORD`' need not be quoted, but it may be. In either event, the output is a quoted string.

Keywords extracted from ignorable commentary at the beginning of a web file are called *global* and are known throughout the code. Distinguish these from *local* keywords extracted from ignorable commentary at the beginning of an include ('`@i`') file. Such keywords are known only during the time that file is being read and are accessible via '`@k`' (see Section 5.5.12 [ATk], page 44).

For convenience, built-ins are defined for some standard RCS global keywords. These are

```
$AUTHOR    => $KEYWORD(Author)
$DATE_TIME => $KEYWORD(Date)
$HEADER    => $KEYWORD(Header)
$ID        => $KEYWORD(Id)
$LOCKER    => $KEYWORD(Locker)
$NAME      => $KEYWORD(Name)
$RCSFILE   => $KEYWORD(RCSfile)
$REVISION  => $KEYWORD(Revision)
$SOURCE    => $KEYWORD(Source)
$STATE     => $KEYWORD(State)
```

There are no such abbreviations for local keywords, because such abbreviations would be expanded during output whereas it is necessary to recognize and expand the local keywords during input. Presumably such local keywords will be used rarely, if at all.

7.2.3.31 `$L`: Change to lower case

'`$L(`*string*`)`' changes *string* to lower case. The argument is first expanded in case it is a macro.

7.2.3.32 `$L_KEYWORD`: Value of local RCS-like keyword

For most purposes, '`$L_KEYWORD`' behaves as '`@k`' (see Section 5.5.12 [ATk], page 44). It is still under development and should not be used yet.

'`$L_KEYWORD("`*Keyword*`")`' extracts (as a character string) the text of an RCS-like keyword defined in the ignorable commentary between '`@z`' and '`@x`' at the beginning of a file included via '`@i`'. '`$L_KEYWORD("`*local keyword*`")`' is expanded during input, and the results are known only during the time the include file is being read.

Note that the argument of '`$L_KEYWORD`' must be a quoted string. For more discussion of the distinction between local and global keywords, please see Section 5.5.31 [ATz], page 50 and Section 7.2.3.30 [$KEYWORD], page 72.

It is expected that local keywords will rarely be used, as fundamental revision-control information should presumably be extracted from the top of the master web file.

7.2.3.33 `$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 this outputs identifiers, not FWEB 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 `$LANGUAGE_NUM` built-in is more useful; see Section 7.2.3.34 [$LANGUAGE_NUM], page 74.

7.2.3.34 `$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 `$IFCASE` construction; see Section 7.2.3.24 [$IFCASE], page 71.

7.2.3.35 `$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.

To expand the argument before taking the length, one can say something like

```
@m $XLEN(s) $LEN(s)
```

7.2.3.36 `$LOCKER`: Value of RCS global keyword `Locker`

Equivalent to '`$KEYWORD(Locker)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.37 `$LOG`: Natural logarithm

'`$LOG(`*x*`)`' returns $\ln x$.

7.2.3.38 `$LOG10`: Logarithm to the base 10

'`$LOG10(`*x*`)`' returns $\log_{10} x$.

7.2.3.39 `$M`: Define a deferred macro

`$M` is equivalent to `$DEFINE`. See Section 7.2.3.12 [$DEFINE], page 69.

7.2.3.40 `$MAX`: Maximum of a list

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

7.2.3.41 `$MIN`: Minimum

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

7.2.3.42 `$MODULE_NAME`: Name of present `web` module

'`$MODULE_NAME`' returns the name of the present `web` module. If the present module is unnamed, it returns the string `"unnamed"`.

7.2.3.43 `$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.

7.2.3.44 `$NAME`: Value of RCS global keyword `Name`

Equivalent to '`$KEYWORD(Name)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.45 `$OUTPUT_LINE`: Current line number of tangled output

This returns the current line number of the tangled output. Contrast this with `$INPUT_LINE`, Section 7.2.3.29 [$INPUT_LINE], page 72.

7.2.3.46 `$P`: The C preprocessor symbol

`$P` is (essentially) a synonym for '`$UNQUOTE("#")`' (see Section 7.2.3.64 [$UNQUOTE], page 78). It is useful for constructing FWEB macro definitions that expand to C preprocessor statements. For example,

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

Another version of the preprocessor symbol is `$PP` (see Section 7.2.3.49 [$PP], page 76). For most purposes, `$P` and `$PP` will behave in exactly the same way. The difference between them is that `$P` is treated as a string (without surrounding quotes), whereas `$PP` is treated as a character. The character nature of `$PP` is used by FORTRAN to reset the column number to 1, so C-like preprocessor commands appear there rather than in column 7.

For further discussion of strings and the differences between `$P` and `$PP`, see Section 7.2.3.1 [Strings and quotes], page 67.

7.2.3.47 `$PI`: Pi

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

7.2.3.48 `$POW`: Exponentiation

'`$POW(`*x*,*y*`)`' generates x^y. (It is a macro defined in terms of `$EVAL` (see Section 7.2.3.17 [$EVAL], page 70) and the exponentiation operator.)

7.2.3.49 `$PP`: The C preprocessor symbol

`$PP` is shorthand for '`$UNSTRING($P)`' (see Section 7.2.3.46 [$P], page 76), or (essentially) a synonym for '`$UNSTRING("#")`' (see Section 7.2.3.65 [$UNSTRING], page 78). It is useful, particularly in FORTRAN, for constructing FWEB macro definitions that expand to C preprocessor statements. For an example, see Section 7.2.3.46 [$P], page 76. For a detailed discussion of the difference between '`$P`' and '`$PP`', see Section 7.2.3.1 [Strings and quotes], page 67.

7.2.3.50 `$RCSFILE`: Value of RCS global keyword `$RCSfile`

Equivalent to '`$KEYWORD(RCSfile)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.51 `$REVISION`: Value of RCS global keyword `Revision`

Equivalent to '`$KEYWORD(Revision)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.52 `$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.

7.2.3.53 `$SECTION_NUM`: Number of current FWEB section

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

7.2.3.54 `$SECTIONS`: Maximum section number

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

7.2.3.55 `$SOURCE`: Value of RCS global keyword `Source`

Equivalent to '`$KEYWORD(Source)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.56 `$SQRT`: Square root

'`$SQRT(`*x*`)`' returns $\sqrt{x}$. It is a convenience macro defined in terms of `$POW`. See Section 7.2.3.48 [$POW], page 76.

7.2.3.57 `$STATE`: Value of RCS global keyword `State`

Equivalent to '`$KEYWORD(State)`'. See Section 7.2.3.30 [$KEYWORD], page 72.

7.2.3.58 `$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.

7.2.3.59 `$STUB`: Trap for missing module

When a missing module is detected, FTANGLE inserts the command '`$STUB(`*module_name*`)`' into the output code. The built-in `$STUB` expands to a function call appropriate to the current language. For example, in C it expands to '`missing_mod`', in FORTRAN it expands to '`call nomod`'.

7.2.3.60 `$TIME`: The time

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

7.2.3.61 `$TRANSLIT`: Transliteration

The macro '`$TRANSLIT(`*s*`, `*from*`, `*to*`)`' interprets each of its arguments as strings (without expanding anything). Then *s* is modified by replacing any of the characters found in *from* by the corresponding characters in *to*. If *to* is shorter than *from*, then the excess characters in *from* are deleted from *s*. As a limiting case, if *to* is empty, then all the characters in *from* are deleted from *s*. For example, '`$TRANSLIT(s, aeiou, 12345)`' replaces the vowels in *s* by the corresponding digits, and '`$TRANSLIT(s, aeiou, )`' deletes all the vowels. The backslash may be used to escape a character, as in ANSI C. For example, '`$TRANSLIT("a\\"\\\\d", "d\\\\a\\"", "D,A'")`' translates into '`A'`,D'. Here one had to explicitly enclose strings involving '`\\"`' in double quotes in order to avoid a complaint about an unterminated string.

7.2.3.62 `$U`: Change to upper case

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

7.2.3.63 `$UNDEF`: Undefine a macro

'`$UNDEF(`*macro*`)`' undefines an FWEB macro.

7.2.3.64 `$UNQUOTE`: Remove quotes from string

'`$UNQUOTE(`*string*`)`' returns *string* without its surrounding quotes. (However, the resulting construction is still treated as a string; no macro expansion is done.)

For a more detailed discussion and a comparison with `$UNSTRING` (see Section 7.2.3.65 [$UNSTRING], page 78), see Section 7.2.3.1 [Strings and quotes], page 67.

7.2.3.65 `$UNSTRING`: Convert string into characters

'`$UNSTRING(`*string*`)`' removes quotes from the string, if they are present, and treats the result as a collection of characters. No tokenization is done, so macro expansion does not operate on those characters.

For a more detailed discussion and a comparison with `$UNQUOTE` (see Section 7.2.3.64 [$UNQUOTE], page 78), see Section 7.2.3.1 [Strings and quotes], page 67.

7.2.3.66 `$VERBATIM`: (Obsolete)

This was an old name for `$UNQUOTE` (see Section 7.2.3.64 [$UNQUOTE], page 78). Please remove all references to this macro from existing codes.

7.2.3.67 `$VERSION`: Present FWEB version number

'`$VERSION`' returns a string built out of the FWEB version number, such as `"1.61"`.

7.2.4 Debugging with macros

If an FWEB macro expands to more than one output line, debugging can be a bit confusing if the debugger (e.g., gdb) displays lines in the web source file instead of the output file (as it normally does for C and C++). While single-stepping through the code, the debugger will incorrectly step the screen display for each output line even if the macro call occupies just one line in the source file. To localize the debugger's confusion, insert a '@#line' command after the macro call. For example,

```
@c
@ Example of a macro that expands to several output lines.
@m UPDATE(i, delta_i)
     i += delta_i;
     store(i)@;
@a
main()
{
UPDATE(j, 5);
@#line
// More code.  The debugger will be in sync from here on.
}
```

An alternative for highly confusing situations is to use the '-#' option (see Section 4.2.78 [-#], page 36).

Another potentially confusing situation occurs when '@%' is used to comment out a line. FWEB deals with the line-number problem that arises here automatically; see Section 4.2.59.6 [-T#], page 31.

FWEAVE makes a valiant attempt to pretty-print (see Section 10.2 [Pretty-printing], page 100) the definitions of both outer macros and FWEB macros in a reasonable way. However, this can be a formidable task, because macro syntax can be essentially arbitrary. Consider, for example, the following definition:

```
@c
@d GET(type) type get_##type()
@a
GET(int){}@; // Expands into 'int␣get_int(){}'.
```

The problem is that the identifier 'type' is used in two different ways: as the type of a reserved word (the second 'type'), and as an ordinary expression (the third 'type'). The first 'type' has both meanings simultaneously. Unfortunately, within any particular language FWEAVE associates one unique type or *ilk* with each identifier.

One solution to this problem is to use the '@R' command (see Section 5.5.23 [ATR_], page 49), which changes the ilk of the very next identifier to integer-like. Thus,

```
@d GET(type) @R type get_##type()@;
```

will format correctly. An alternative solution uses the related command '@E', which changes the ilk of the very next identifier to an ordinary expression. Thus,

```
@f type int
@d GET(type) type get_##@Etype()@;
```

Other types of troublesome situations involve spaces. When FWEB understands the syntax, it inserts spaces automatically to make the output pleasing. Consider, however, the (somewhat contrived) example

```
@c
@d A(x, y) x y
@d B s1;
@d C s2;
@a
A(B, C)@;
```

Here FWEAVE will consider '`x`' and '`y`' to be ordinary identifiers (simple expressions), and will abut them with no intervening spaces, which is confusing to read. The solution is to insert a space manually with '`@,`':

```
@d A(x, y) x @, y
```

(Whether one should write macros like this at all is a separate issue.) For a related example, see the discussion of Section 5.12 [ATcomma], page 55.

7.3 Preprocessing

Generally, the FWEB preprocessor commands follow a syntax identical to their C/C++ counterparts. The one exception is the '`@#line`' command. Whereas the C command takes a line number and file name as arguments, the FWEB command takes no arguments; its expansion automatically inserts the current line number and file name. This command should be necessary only in rare circumstances. One of those involves situations in which an FWEB macro expands to more than one output line; see Section 7.2.4 [Debugging with macros], page 79.

The 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, see Section 7.2 [FWEB macros], page 63.

The syntax of each command is as follows:

`@#line` — Insert a `#line` command.

`@#define` *identifier*
: — Define an FWEB macro; equivalent to '`@m`'.

`@#undef` *identifier*
: — Undefine an FWEB macro.

`@#ifdef` *identifier*
: — Is FWEB macro defined? Equivalent to '`@#if defined identifier`'.

`@#ifndef` *identifier*
: — Is FWEB macro not defined? Equivalent to '`@#if !defined identifier`'.

`@#if` *expression*

`@#elif` *expression*

`@#else`

```
@#endif
```

In the '`@#if`' statement, the *expression* may contain FWEB 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

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 an FWEB macro; to 0 (false) otherwise. The construction '`@#if defined A`' works the same way as '`@#ifdef A`', but one 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 FWEB 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, $\tilde{}\texttt{0} = -1$.

Binary operators:

`^^` — Exponentiation (all languages). $2\hat{}\hat{}3 = 8$.

`^`, `**` — 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 is the exclusive-or operator for C. Also, note that the bitwise operators should almost never be used. For logic, almost always one will be using '`==`', '`!=`', '`&&`', and '`||`'.

8 LANGUAGES

FWEB has the ability to work with more than one source 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 (deliberately) 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 12.3.8 [Miscellaneous params], page 118.

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 the option '`-`*l*`text`', 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`

An example of a command with the optional *text* field is '`@n/`'. By definition, this is equivalent to '`@n[-n/]`'. Thus, it both sets the language and invokes a command-line option.

As another example, '`@n9`' really means '`@n[-n9]`'. Thus the language is first set to FORTRAN, then reset to FORTRAN-90. One doesn't need to worry about this detail.

`@n9[-n&]`

means set the language to FORTRAN–90 and use free-form syntax with the ampersand as the continuation character. (This construction is now FWEB's default.)

The brackets may contain more than one space-delimited option.

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. For historical reasons, the default language is FORTRAN-77, but *do not rely on this; always include a language command.*

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—e.g., 'test.f' and 'test.c'. Particularly note that one did not need an '@c' command in the last section because the language was C when '@<C@>' was first encountered.

8.2 Special hints and considerations for each language

One important thing to keep in mind is that in FWEB an identifier may have, for each language, precisely one meaning throughout the document. This restriction is not necessarily in accord with the syntaxes of the various source languages. See, for example, the discussions in Section 8.2.2 [Cpp], page 85 and Section 8.2.3 [Fortran], page 85.

8.2.1 Special considerations for C

- FTANGLE treats the construction '0b...' as a binary notation that it expands to an unsigned decimal number. Thus, '0b101' expands to 5 and '0b1111111111111111' expands to 65535.
- FWEAVE processes **typedef** statements during phase one, so they will format properly even if they are used in a documentation part before they are defined in a code part.
- The '-H' option helps one to deal with identifiers defined in header files. See Section 4.2.17 [-H_], page 20.
- Note that in C structure and enum tags do not define a new type, so the tag name does not get highlighted in boldface, underlined in the index, etc. (That is, if one says

'struct S {...};', one can't say 'S s;', one must say 'struct S s;'.) This is a good reason for using C++, where such tags do define a new type.

(To be completed.)

8.2.2 Special considerations for C++

- All of the items in the previous section (see Section 8.2.1 [C], page 84) still apply.
- The '@{' command is very useful for beautifying very short definitions of member functions such as constructors. See Section 5.9 [ATlb], page 53
- Essentially, FWEAVE has only one name space, global to the entire code; those names do not obey any concept of scope. In various situations in C and C++, however, multiple namespaces are used, or the interpretation of a name changes according to its scope. Thus, the design of FWEAVE imposes a few restrictions on one's programming style. (Remember, FWEAVE doesn't know nearly as much as a language compiler.)

  One example in C++ has to do with formal types in templates. Consider the following example:

  ```
  template <class Type>
  class A
  {
  private:
          Type *p;
  }
  ```

  In order that the class definition be typeset correctly, 'Type' must be understood to be a reserved word like **int**, and that is correctly figured out by the first **class** command. However, according to C++, the scope of 'Type' is local to the class definition; unfortunately, FWEAVE does not respect that locality and will always treat 'Type' as an **int** from the point of the 'class Type' construction to the end of the source code. Thus, one should use such dummy variables as 'Type' only as formal template parameters, never as ordinary variables.

8.2.3 Special considerations for FORTRAN

8.2.3.1 Items for both FORTRAN-77 and FORTRAN-90

- FTANGLE will translate into unsigned decimal numbers the binary notation '0b...', the octal notation '0...', and the hexadecimal notation '0x...'. Thus, '0b101' expands to 5, '0101' expands to 65, and '0x101' expands to 257.
- Don't use the column 1 'C' commenting convention. Use '/* ... */' or '// ...'.
- For compiler directives, use '@?' (see Section 5.8.4 [AT?], page 53), not a 'C' in column 1.
- If you are going to use the recommended '// ...' convention for short comments, you must say '@n/' (see Section 4.2.40 [-n/], page 27) or '@n9[-n/]' as your language command. Otherwise, \FWEB\ will treat the '//' as \FORTRAN's standard token for concatenation. (You may always use '\/' for concatenation.)

- If you want to completely comment out a whole block of code, use the preprocessor construction '`@#if 0...@#endif`' (see Section 7.3 [Preprocessing], page 80). Don't put a comment character at the beginning of each line; that prevents FWEAVE from formatting the code sensibly and can be annoying to undo. With the preprocessor form, one can also implement conditional comments by using FWEB preprocessor macros: e.g., '`@#if(DEBUG)...@#endif`'.

  Pre-FWEB codes may have such blocks commented out with a '`C`' in column 1. Those should be converted to the preprocessor construction. However, if you're in a real hurry, temporarily use the '`-nC`' option (see Section 4.2.36 [-nC], page 25) to kill those lines very early in the processing, before they can give you all kinds of trouble.
- An unfortunate byproduct of using '`//`' for short comments is that, in general, format constructions like `format(//)` won't work. (It will work if one uses '`-nC`'; see Section 4.2.36 [-nC], page 25.) Alternatively, one can say `format(/ /)`.
- Consecutive lines commented out with a '`C`', '`c`', '`*`', or '`!`' in column 1 are converted into a single comment before processing by FWEB. Large blocks of such lines (common in pre-FWEB code) may overflow FWEB's tables. To avoid that, insert blank lines between some of the comments. Better, however, is to move most such blocks out of the code part to the TeX part of the section. It's most readable to have only a few very short comments interspersed in the code.

  To help with conversion of existing codes, the command-line option '`-nC`' can be used to completely ignore comment lines.
- '`@`' commands should, by and large, start in column 1. That's not necessary for short module names that fit on one line. However, a long module name that must be broken across lines must begin in column 1, as in

  ```
  @n
  @
  @a
  @<This is a module name
  broken across lines@>@;
  ```

  Failure to do this results in a spurious semicolon being inserted in the middle of the name. This happens because the FORTRAN-77 input driver does various low-level manipulations of the source before it presents it to the innards of FWEB; it's not tokenizing the source at that time and doesn't understand all of the FWEB syntax such as module names.
- Define symbolic statement labels with '`#:0`' (see Section 7.2.2 [Tokens], page 65). Such names should be followed by a colon. Thus,

  ```
  @n
  @
  @m EXIT #:0
  @m ABORT #:0
  @a
  .
  .
  ABORT: continue
  .
  ```

```
      .
      EXIT: continue
      .
      .
```

- By default, statement labels are `\llap`'d from the body of the statement. With this convention, long labels can extend too far into the left margin. Instead, try the command-line option '`-n:`' (see Section 4.2.34 [-ncolon], page 25), which puts them on a separate line. Alternatively, one can redefine the macro `\Wlbl`, found with some discussion in '`fwebmac.sty`'.
- As a suggestion, use upper case for I/O keywords such as `IOSTAT`. However, by default the lower-case forms are also recognized. To permit only upper case, use '`-k`' (see Section 4.2.23 [-k], page 22). Note that although there is a command '`-nk`', it is unfortunately not related to '`-k`'.
- One may use '`^`' as an alternative for the exponentiation operator '`**`'.
- FWEB attempts to be helpful and tries to expand the operators '`++`', '`--`', '`+=`', '`-=`', '`*=`', and '`/=`' in a way compatible with the usage in C and C++. For example, it expands '`x += y`' into '`x = x + (y)`'. This feature can be a great time-saver and also makes the code substantially more legible; it is strongly recommended. To turn off this feature, use the option '`-+`'. See Section 4.2.79 [-plus], page 37.

  Notice that in FORTRAN-90 '`/=`' is a token for "not equal," so if you want to use that you must use the '`-+`' option. However, a better solution is to use '`!=`', FWEB's preferred operator for "not equal."
- By default, the operators `.true.` and `.false.` will weave as caligraphic T and F. That appearance be changed by redefining the macros `\WTRUE` and `\WFALSE` in '`fwebmac.sty`' or in the limbo section of your source file.
- If FTANGLE messes up and outputs incorrect FORTRAN code, try tangling with the command-line option '`-#`' (see Section 4.2.78 [-#], page 36) (and then report the problem.)

8.2.3.2 Items specific to FORTRAN-77 and fixed-form FORTRAN-90

- By default, when processing the code part the FORTRAN driver inserts semicolons automatically at the end of each logical statement. Thus, the core of FWEB is presented with a uniform syntax. However, when one escapes into code mode by using vertical bars, those semicolons aren't inserted, so something that appears a first glance to be complete statement may not be formatted as one might expect. Thus, the construction '`|5: continue|`' doesn't format quite properly (the colon disappears); this problem is solved by putting a semicolon after the '`continue`'. Also, if one is talking about multiple statements (for example, with a shift into code mode during TeX documentation), there's no choice but to insert the semicolon between statements. For example, '`|a = b; c = d;|`'.

8.2.3.3 Items specific to FORTRAN-90

- If FORTRAN-90 is selected (see Section 4.2.31 [-n9], page 24), the default is *free-form* syntax (lines are continued by a trailing ampersand). However, automatic line breaking is done in a way compatible with fixed-form syntax as well.
- With free-form syntax, comment lines in the tangled output file begin with '`!`'. But such lines are not recognized on input unless '`-n!`' is used. See Section 4.2.41 [-n!], page 27.
- Beginning with Version 1.61, by default (pseudo-)semicolons are automatically inserted in free-form \Fortran-90 code, as one would expect. For more discussion, see Section 4.2.32 [-nAT;], page 24 and Section 4.2.33 [-n;], page 24.

(To be completed.)

8.2.4 Special considerations for RATFOR

For some warnings about RATFOR, see Section 9.3 [Caveats], page 90.

8.2.5 Special considerations for TeX

'`@Lx`' is supported only to the extent that `fwebmac.sty` can be generated correctly from `fwebmac.web`. You are welcome to experiment, but you may encounter difficulties (which you should report; see Chapter 15 [Support], page 131).

(To be completed.)

8.2.6 Special considerations for the VERBATIM language

Unfortunately, the VERBATIM language is not fully debugged. Therefore, it is not recommended for general use. (To be completed.)

9 RATFOR

"RATFOR" stands for "RATional FORtran." It endows FORTRAN with a C-like syntax. Certain loop and other constructions (such as '`switch`' or '`i++`') that are not allowed in FORTRAN are allowed in RATFOR; FWEB translates those into proper FORTRAN.

Although RATFOR is a definite improvement over FORTRAN, it certainly does not have the power of C (e.g., elegant pointer notation) or C++ (e.g., classes). Many advantages accrue by taking the time to learn C. RATFOR offers a gentle transition. (It is not supported very actively any more.)

9.1 RATFOR syntax

A sample RATFOR program is

```
@r
@
@a
program main
{
integer k;
real fcn, x;

for(k=0; k<10; k++)
        {
        x = fcn(k);

        if(x < 0.0)
                {
                x = 0.0;
                break;
                }
        }
}
```

The concluding brace of a function is translated into an **END** statement. Note the use of semicolons to terminate statements, braces to delimit compound statements, '`<`' instead of '`.LT.`', the C-like **for** construction, and the '`k++`' expression.

Constructions like '`k++`' or '`k -= l + 1`' must be used with great care. They translate to statements involving '`=`' signs, so they can be used only where simple statements are allowed, not essentially anywhere as in C (for example, they cannot be used as function arguments).

9.2 RATFOR commands

9.2.1 RATFOR–77 commands

```
break; // Used with case or to break out of loops, as in C.
case i: // Used with switch.
default: // Used with case, as in C.
do ...; {...} // Note the semicolon (unnecessary if followed by a compound stmt).
else {...} // Used after if as in C.
for(a;b;c) {...} // As in C.
if(condition) {...}
next; // Equivalent to C's |continue| statement; go to bottom of loop.
repeat {...} until(condition); // Equivalent to C's do␣{...}␣while();
return expression; // As in C.
switch(expression) {...} // As in C.
while(condition) {...} // Like C's while.
```

9.2.2 Additional RATFOR–90 commands

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

9.3 Caveats 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. (Use symbolic statement labels instead; see the discussion of '`#:0`' in Section 7.2.2 [Tokens], page 65.)
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 RATFOR program unit; it is added automatically by FWEB when the closing brace is sensed.

10 DOCUMENTATION

FWEB uses LaTeX to produce its documentation. Plain TeX is no longer supported.

It is not necessary to be very familiar with LaTeX in order to use FWEB effectively. FWEB does complicated things behind the scenes, relieving the programmer of many burdens. If you don't need complicated mathematics, one needs to know virtually no LaTeX at all in order to document a section of code. And if you do need to typeset math, consider that LaTeX makes this daunting task about as simple as one could hope.

If you're an FWEB beginner, don't bother diving into the details of this section until you really need to.

10.1 Typesetting

FWEB's "new look" (beginning with version 1.40) is designed to work only with LaTeX. The new look is more book-like, following ideas from Briggs' `nuweb`. By default, it uses default LaTeX section numbers such as 1.5.32; however, sections may be numbered with consecutive integers by specifying the LaTeX2e package `fwebnum`; see Section 10.1.6 [Numbering], page 98.

10.1.1 FWEAVE's OUTPUT

When one says '`fweave test`', the file '`test.tex`' is created. Some TeX commands contained in this file are created automatically; others are copied from the web source file. They are organized into several sequential groups, as follows.

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

   By default, the initial input command is '`\input fwebmac.sty`' (see Section 10.1.2 [fwebmac.sty], page 92). The name of the macro package can be changed with the '`-w`' command-line option, but that is dangerous and useful only for very special effects. See Section 4.2.66 [-w], page 34.
2. `\Wbegin` command.

   The `\Wbegin` macro sets up certain defaults (which can be overridden in the limbo section). In LaTeX, it also issues the '`\documentclass{article}`' and '`\begin{document}`' commands.
3. Limbo text from the style-file parameter `limbo.begin`. See Section 12.3.8.11 [S_limbo], page 120.
4. Limbo text from '`@l`' commands. See Section 5.5.14 [ATl], page 45.
5. User's limbo section.
6. Limbo text from the style-file parameter `limbo.end`. See Section 12.3.8.11 [S_limbo], page 120.
7. TeX commands for individual WEB sections.
8. `\input` command to read in the index data file.
9. `\input` command to read in the module-list data file.
10. `\Winfo` command (summarizes some status information).
11. `\Wcon` command (generates the Table of Contents, and ends the run).

10.1.2 The macro package '`fwebmac.sty`'

FWEAVE works in conjunction with the macro package '`fwebmac.sty`', which is always read into the '`.tex`' file by default. This file is (overly) complicated, so one should not mess with it unless in dire emergency. Most of its commands are intended for behind-the-scenes processing. However, some features may be of general interest; these are described in the items below.

For the most part, macros used internally by '`fwebmac.sty`' begin with an uppercase '`W`'. If you are worried about macro conflicts, a complete list of the macros appearing in '`fwebmac.sty`' can be found in the Index produced by weaving '`fwebmac.web`'.

10.1.2.1 User macros

For the user's convenience, '`fwebmac.sty`' defines a variety of macros such as '`\FWEB`', '`\Fortran`', etc. Refer to '`fwebmac.web`' for a complete list.

FWEAVE usurps various common single-character macros such as '`\.`' for its own purposes. So the user can still access their original definitions, those are '`\let`' equal to alternative commands such as '`\period`'. For example, commands such as the following are executed in `fwebmac.sty`:

```
\let\amp\&
\let\at\@@
\let\bslash\\
\let\caret\^
\let\dollar\$
\let\dstar\*
\let\equals\=
\let\leftbrace\{
\let\period\.
\let\rightbrace\}
\let\vertbar|
\let\PM\#
\let\PC\%
```

(Some of the more inscrutable synonyms are for historical reasons.)

For the most up-to-date and detailed information, refer to '`fwebmac.web`'.

10.1.2.2 Fonts

Several fonts have been declared. Those include

- '`\titlefont`' (large sans serif),
- '`\ttitlefont`' (large typewriter),
- '`\SC`' (small caps),
- '`\Csc`' (Caps/small caps), and
- '`\tentex`' (TEX's extended character set).

For illustrations and further details, see '`fwebmac.web`'.

To typeset a string of characters in typewriter type, one may use the '`\.`' macro. (More precisely, the name of this macro is the value of the style-file parameter `format.typewriter`. For more information, see Section 12.3.5.1 [S_format], page 115.) When using this, one must escape the special characters '` \#%$^_{}~&`', as in '`\.{\\alpha}`'. (FWEAVE does that escaping automatically when typesetting strings in code mode.) You may wish to surround '`\.{...}`' with an '`\hbox`'; that is not done by default because FWEAVE uses special trickery to break long strings in code mode automatically, and that breaking would be inhibited by an '`\hbox`'.

10.1.3 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 (and Plain TeX is *not* supported); version 1.50 supports LaTeX2e. LaTeX's `\output` routine is used, as are its sectioning commands (with minor changes), Table-of-Contents facilities, etc.

The following discussion is based on LaTeX2e. If LaTeX2e is not installed, FWEAVE recognizes that fact and issues the '`\documentstyle`' command instead of '`\documentclass`'.

Users are strongly encouraged to upgrade to LaTeX2e. A useful book that describes the present state of LaTeX is Goossens, Mittelbach, and Samarin, *The LaTeX Companion* (Addison–Wesley, Reading, MA, 1994).

10.1.3.1 LaTeX's document class

An FWEB/LaTeX document is set up with the '`\Wbegin`' command, issued automatically by FWEAVE. See the summary at the end of this section for the essence of what the '`\Wbegin`' command accomplishes.

FWEAVE uses `\documentclass{article}` by default. In principle, the document class can be changed by the FWEB style-file option '`LaTeX.class`'; see Section 12.3.5 [Fwebmac params], page 115. However, FWEAVE *has not been tested with most other document classes*. It will probably not work with most document classes that redefine the sectioning commands from those of `\documentclass{article}`. However, it *may* work with the `revtex` scientific macro package. See Section 10.1.3.2 [REVTeX], page 94.

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

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

To get two-sided printing, for example, one would say

```
LaTeX.class.options "twoside"
```

To specify user packages—i.e., to obtain the effect of '`\usepackage[pkgoptions]{pkgname}`'—use the style-file parameters `LaTeX.package` and `LaTeX.package.options`, as in

```
LaTeX.package "pkgname"
LaTeX.package.options "pkgoptions"
```

For example, to indent the first line of every section and to permit the use of the `multicol` package (the latter is a useful way of substantially cutting down on white space), say

```
LaTeX.package "indentfirst,multicol"
```

Note that specifying `LaTeX.package` and `LaTeX.package.options` results in the execution (by the `\Wbegin` macro) of precisely *one* line of the form

```
\usepackage[myoptions]{mypackages}
```

Sometimes one instead needs to have multiple `\usepackage` lines, such as

```
\usepackage[option1]{package1}
\usepackage[option2]{package2}
```

To get this effect, one can put these commands explicitly into the style-file parameter `doc.preamble` (see discussion two paragraphs below), as in

```
doc.preamble = "\\usepackage[option1]{package1}\
                \\usepackage[option2]{package2}"
```

TEX commands in the user's limbo section of the `web` source file will be processed *after* the `\begin{document}` command. Limbo commands from the style file can be inserted before and/or after those in the limbo section with the aid of the style-file parameters '`limbo.begin`' and '`limbo.end`'; see Section 12.3.8.11 [S_limbo], page 120.

If there is a compelling reason to insert one's own LaTEX commands between the '`\usepackage`' and '`\begin{document}`' commands, one may use the style-file parameter '`doc.preamble`', whose value is a string consisting of LaTEX commands (empty by default). Those commands are processed immediately before '`\begin{document}`'. One use of '`doc.preamble`' is to inhibit FWEB's tendency to keep a section together on one page. To make it break more readily in the middle of sections (particularly useful for multicolumn output), say

```
doc.preamble "\\secpenalty=0"
```

In summary, the beginning of the file output by FWEAVE looks like the following, where '`<parameter>`' means the contents of the style-file string called '`parameter`':

```
\input fwebmac.sty
\Wbegin{many obscure arguments}
<limbo.begin>
Optional TeX commands copied from user's limbo section
<limbo.end>
```

The '`\Wbegin`' command essentially does the following:

```
\documentclass[<LaTeX.class.options>]{<LaTeX.class>}
\usepackage[<LaTeX.package.options>]{<LaTeX.package>}
<doc.preamble>
\begin{document}
```

For precise information about how '`\Wbegin`' works, see `fwebmac.web`. If you feel that macro absolutely needs to be changed, please inform the developer (see Chapter 15 [Support], page 131).

10.1.3.2 Using REVTEX

REVTEX is the standard macro package used for formatting scientific papers submitted to the American Physical Society, the American Institute of Physics, and some European journals. It modifies the sectioning commands of `\documentclass{article}` and provides various other useful macros.

Unfortunately, as of August, 1998, REVTEX is not fully compatible with LaTEX2e; it must be invoked with `\documentstyle{revtex}`, not `\documentclass`. This is annoying, because FWEB's macros in '`fwebmac.sty`' default to `\documentclass` if they recognize that LaTEX2e is loaded.

To use REVTEX, uncomment the line in '`fwebmac.sty`' that says `\useREVTeXtrue`. (One cannot say '`\useREVTeXtrue`' in the limbo section of one's `web` source, because the document class has already been selected by that time.) You may wish to rename the resulting file, say to '`rwebmac.sty`', so it can be loaded in place of the standard '`fwebmac.sty`'. To do that, one would use the command-line option '`-wrwebmac.sty`' (see Section 4.2.66 [-w], page 34).

Saying `\useREVTeXtrue` selects `\documentstyle` rather than `\documentclass`. To implement a standard command such as `\documentstyle[aps,my_macros]{revtex}`, use the style-file ('`fweb.sty`') parameters `LaTeX.style` and `LaTeX.options`, as in

```
LaTeX.style "revtex"
LaTeX.options "aps,my_macros"
```

Here '`my_macros.sty`' would be a user's macro package loaded in addition to those of REVTEX and FWEB.

REVTEX support is extremely recent. There may be glitches; please report those. In a pinch, if LaTEX stops while processing a REVTEX file produced by FWEAVE, try typing 's' (scroll mode) to force it to continue; you might get usable output.

10.1.3.3 LaTEX packages related to FWEB

The following packages are supplied with the FWEB distribution and can be used to achieve special effects. Packages are invoked by giving their names as arguments to the `LaTeX.package` command; see Section 12.3.5.3 [S_LaTeX], page 116.

- `fwebinsert` — Enables insertion of woven code into a LaTEX document. See Section 10.1.6.1 [Inserting woven code], page 99.
- `fwebnum` — Number each section in ascending integer order. See Section 10.1.6 [Numbering], page 98.
- `idxmerge` — Merge several stand-alone indexes. See Section 11.3 [Merging indexes], page 105.

10.1.3.4 Sections in LaTEX

FWEB's sectioning commands '`@*`' and '`@*`n' 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 untitled '`@␣`'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 previous scheme means that it may be necessary to run LaTEX as many as three times in order to resolve all sectioning and cross-reference information correctly. You should be warned in such cases. If not, you will recognize difficulties by noting that the Table of Contents or section numbering is incomplete.

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.

A discussion of alternative section-numbering schemes is given in Section 10.1.6 [Numbering], page 98.

10.1.3.5 LaTeX's index.

The Index should be the last section of the code, and should be begun by the command '`@* \INDEX.`'. For more information, see Section 12.3.1.1 [S_index], page 113.

The challenge of typesetting the Index is to get it into two-column mode in the best possible way. In the original Plain-TeX FWEB, special code was provided for this. With LaTeX, however, one wants to use standard features.

The best solution is to use the user package `multicol`. If that is loaded by means of the style-file statement '`LaTeX.package "multicol"`', then any text typed by the user following the '`@* \INDEX.`' command will be typeset in single-column mode, after which two-column mode is entered. If it is not loaded, a '`\twocolumn`' command is issued *before* the index section is begun (in order to get the Index started on a new page).

More precisely, what happens is the following. When the '`@* \INDEX.`' command is recognized, essentially the following operations are performed, where the results are bracketed in the form '`[multicol : nomulticol]`':

```
\beforeindex [\newpage : \twocolumn]
[print INDEX section heading]
\startindex  [\begin{multicols}{2} : \medskip]
\Wfin        [\end{multicols} : \relax]
```

(Use of the asymmetrical name '`\Wfin`' is for historical reasons.)

The positioning of '`\beforeindex`' suggests a way of printing the entire document in two-column mode. If one enters multi-column mode in the limbo section, then '`\beforeindex`' can be used to terminate it. It is best to do this at the *end* of the limbo section; otherwise user macro definitions in the limbo section must be made `\global` in order that they remain defined in the Index. The relevant commands can be placed in the style file:

```
LaTeX.package "multicol"

doc.preamble "\\secpenalty=0"

limbo.end "\\def\\beforeindex{\\end{multicols}\\newpage}\n\
\\begin{multicols}{2}\n\
\\raggedcolumns"
```

Just to repeat, use only the first command to get just the Index printed in two-column format; use the second and third ones to make the entire document two-column.

10.1.3.6 LaTeX's Table of Contents

LaTeX uses the 'aux' file to accumulate the information for the Table of Contents.

When LaTEX is used, the Table of Contents appears at the front of the document by default (beginning with version 1.61). This is accomplished by setting the default value of the style-file parameter `limbo.end` to `"\\FWEBtoc"`, where `\FWEBtoc` is defined in '`fwebmac.sty`'. If you initialize `limbo.end` yourself in '`fweb.sty`', you should include `"\\FWEBtoc"` at the end of that initialization if you want the Table of Contents to appear in the beginning. Otherwise, it will appear at the end.

In essence, the Table of Contents is produced by the LaTEX commands

```
\pagenumbering{roman}
\maketitle
\topofcontents
\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 it is better to use FWEB's `\Title` macro instead of `\title`:

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

By default, the argument of the `\Title` macro is printed both on the title page and as a running headline in the document. The default font for the title is `\ttitlefont`; that for the running headline is `\large\tt`. However, `\Title` has one optional argument that allows one to override the running headline, perhaps by specifying a shorter form. Say

`\Title[`*Short title*`]{`*Long title*`}`

to make the running headline be '`\large\tt` *Short title*' and the title-page title be '`\ttitlefont` *Long title*'.

The `\FWEB` `\Title` macro calls LaTEX's `\title` macro with the long title as its argument. By default, FWEAVE uses (in the '`\Wbegin`' macro)

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

Section numbers in the Table of Contents are produced by the LaTEX macro `\numberline`. LaTEX's default definition is inadequate when section numbers are very large; they extend to the right and can overwrite the section name. The macro is redefined more appropriately when the package `fwebnum` (see Section 10.1.6 [Numbering], page 98) is used.

10.1.3.7 Customizing LaTEX's output

Several (TEX) flags are provided to change the appearance of the final LaTEX document. (This appearance is a bit experimental, and it is fair to say that not everything may be fully debugged; please report problems.) These are ('`...`' means either '`true`' or '`false`')

- `\pagerefs...` (index references by pages or section numbers);
- `\numberTeX...` (number the beginning of unnamed TeX parts);

- `\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
```

If desired, one may override these in the limbo section. (They are defined using Plain TEX's '`\newif`' rather than the equivalent LaTEX command because they may also be used when LaTEX is not present.)

`\numberTeX` is on the verge of obsolescence. Try to not use it; never use it in conjunction with the package `fwebnum`. See Section 10.1.6 [Numbering], page 98

10.1.4 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`'. (Presently, this will not work correctly when `multicol` is used for the body of the document.)

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

10.1.5 Page headers

The very top (header) line on each page of FWEAVE's output contains several pieces of information:

- the current section name or document title;
- the page number;
- the range of LaTEX section numbers on the page (these are preceded by the § symbol); and
- the range of integer section numbers as understood internally by FWEAVE (those are in square brackets and preceded by the '`#`' sign).

10.1.6 Section numbering schemes

The FWEB commands '`@*`' and '`@␣`'are translated by complicated magic into LaTEX commands such as `\section`, `\subsection`, etc. By default, use of `\documentclass{article}` then produces Dewey-decimal section numbers such as 2.13.4 (subsubsection 4 of subsection 13 of section 2). When the section tree is very deep, these numbers can look somewhat obtrusive.

An alternative scheme (that of the original WEB) is to merely number each section in ascending integer order, beginning with 1. This can be done by specifying the package `fwebnum`, as in

```
LaTeX.package = "fwebnum"
```

This package is supplied with the FWEB distribution; it is still somewhat experimental.

By default, `fwebnum` numbers all sections, including unnamed ones. To prohibit numbering of unnamed sections, use the package option `dontnumberunnamed`, as in

```
LaTeX.package.options = "dontnumberunnamed"
```

This option will eventually make `\numberTeX` obsolete; do not use `\numberTeX` in conjunction with `fwebnum`.

10.1.6.1 Package `fwebinsert`: Inserting FWEAVE's output into a LaTeX document

Beginning with version 1.61, it is (barely) possible to insert the TeX output woven by FWEAVE into a LaTeX document. For example, a code listing could be an appendix to a dissertation, or a handbook on numerical methods could insert fragments of code formatted by FWEAVE.

Suppose one has the file '`test.web`' and used FWEAVE to create '`test.tex`'. Unfortunately, it does *not* work to simply `\input test.tex` into a LaTeX document, because by default '`test.tex`' operates in a "stand-alone" mode and tries to issue a `\begin{document}` command.

Instead, one must use the package `fwebinsert` and the special input command `\FWEBinput`, as in the following example. There are two important steps.

1. Use FWEAVE to create '`test.tex`'. [You may wish to use the '`-x`' flag (see Section 4.2.67 [-x], page 34) to prevent some of the lists at the end, such as the index or module list, from being printed.]
2. Now '`latex test`' until all of the section numbering is up-to-date. (This step is necessary because information in the '`aux`' file is used in processing the section headings.)

Now '`test.tex`' is ready to be inserted in a code like the following:

```
\documentclass{article}
\usepackage{fwebinsert}

\begin{document}

\section{Body}

The body of the document.

\appendix

\FWEBinput{test}

\end{document}
```

Note that the '`@*`' commands in '`test.web`' are converted into LaTeX sectioning commands such as `\section`. The above example works correctly because the first '`@*`' in '`test.web`' is equivalent to a `\section` (level 0) command, which should indeed immediately follow an `\appendix` command. Suppose, however, that you wanted to input '`test.web`' as part of the body of the above example, and wanted the '`@*`'s to be treated as subsections (level 1) rather than sections. To tell `fwebinsert` what level number to assign to the '`@*`'s, provide that number as an optional argument to `\FWEBinput`, as in the following example:

```
\documentclass{article}
\usepackage{fwebinsert}

\begin{document}

\section{Body}

The body of the document.

\FWEBinput[1]{test}

\end{document}
```

Alternatively, say `\FWEBlevel{1}` before the `\FWEBinput`. (The optional argument construction merely calls `\FWEBlevel`.)

Here are some caveats about `fwebinsert`:

- Implementing this package was tricky. It may work in simple circumstances, but it is not fully debugged.
- The `\FWEBinput` command surrounds the included TEX code with `\begingroup`...`\endgroup`, in an attempt to prevent various macro conflicts. As it stands, the command `\fwebinput` is `\let` equal to `\FWEBinput`. If necessary, one could redefine `\fwebinput` to not include the enclosing `\begingroup`...`\endgroup`.
- For anything except level-0 inclusions, one should have just one `\FWEBinput` command following each sectioning command. (This is a bug.)
- One is supposed to be able to use the package `fwebnum` (see Section 10.1.6 [Numbering], page 98) in conjunction with `fwebinsert`. One can apply that to either the included file (via a `LaTeX.package` entry in '`fweb.sty`'), the including file (via a `\usepackage` command), or both. Try out these various combinations to see what emerges.

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`').

Pretty-printing is one of those topics that can arouse strong passions: your idea of what's esthetic may not be mine. Unfortunately, FWEB's formatting rules are mostly hard-coded, so if, for example, you don't like the way braces are arranged in typeset C code, you're mostly stuck. Most directly, this possibly undesirable choice comes from design decisions made by previous authors. It also makes FWEAVE very fast, and enables certain complicated tricks that seem difficult or impossible to accomplish with a completely customizable approach. The latter seems quite formidable, and has not been attempted—a good thesis project for the 21st century.

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; they are ignored by FTANGLE. The pseudo-operators are

`@e` — pseudo-expression. See Section 5.13 [ATe], page 57.
`@;` — pseudo-semicolon. See Section 5.13.2 [AT;], page 58.
`@:` — pseudo-colon. See Section 5.13.3 [ATcolon], page 59.

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 one 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.`	`(||)`	$\to \vee$
`.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.`	`(&&)`	$\to \wedge$	`//`	`(\/)`	$\to \|$

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, in the opinion of the author 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 compiler language itself 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 if one wants to include a backslash one must escape it with another backslash. For example, one 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 (maybe it will be someday). In the meantime, one has the command '`@W`', which provides a restricted form of such a facility. *This command 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. (However, one may need some rather intimate understanding of FWEAVE's output in order to ensure that things always work correctly.)

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" (sorry, that isn't here yet), where an example is given.

11 FWEB's INDEX.

FWEB has several powerful indexing facilities:

1. It sorts and writes its own self-contained (*internal*) index, including cross-references to all the variables as well as items inserted by the user.
2. It can write its cross-reference information to a file formatted for use by the `makeindex` utility. This feature facilitates creation of a master index that contains information about several `web` files.

11.1 FWEB's self-generated index

One of the most useful features of FWEB is that it automatically generates an Index of all variable usage. One can also insert one's own index entries by using the commands

- '`@^`' (entry in Roman type; see Section 5.10.4 [AT^], page 54),
- '`@.`' (entry in typewriter type; see Section 5.10.5 [ATdot], page 54), and
- '`@9`' (user-defined format; see Section 5.10.6 [AT9], page 54).

(More discussion to be completed.)

11.2 Creating a stand-alone index with `makeindex`

In addition to the internal index described in the previous section (see Section 11.1 [Internal index], page 103), FWEAVE can write the index data to a file formatted for later, stand-alone processing by the `makeindex` utility. (Several such indexes can be merged together; see Section 11.3 [Merging indexes], page 105.) The procedure is simple, although the following discussion goes into some rather arcane details.

11.2.1 Creating a stand-alone index: Summary

As a quick reference for those who have already read the details in the next subsection, the procedure to print a stand-alone index with `makeindex` is as follows. First, create, if necessary, a file '`index.tex`' that `\inputs` '`index.ind`'. (A skeleton is illustrated in the next subsection.) Then:

```
fweave -XI test.web % Creates test.idx and test.sty.
makeindex -s test.sty -o index.ind test.idx % Creates index.ind.
latex index
```

If you're not happy with the `\pg` macro supplied in '`fwebmac.sty`', define it yourself in '`index.tex`'.

In this procedure, note the use of the '`-XI`' option and the use of a different root name ('`index`' here) for the output file.

11.2.2 Creating a stand-alone index: Details

To create an index file in a form suitable for later stand-alone processing by `makeindex`, use the '`-XI`' option to FWEAVE. If the `web` file is '`test.web`', the default name of the `makeindex` output file is '`test.idx`'. (This name can be overridden by the style-file parameter `makeindex.out`.) Run `makeindex` on '`test.idx`' to create the LaTeX file '`index.ind`' (see following discussion for details). A stand-alone index can then be produced by saying '`latex index`', where a skeleton version of '`index.tex`' would be

```
% index.tex --- skeleton for printing a stand-alone index.
\documentclass{article}
\usepackage{fwebmac}

\begin{document}

\input{\jobname.ind}

\end{document}
```

(In practice, a more involved procedure will probably be followed; see below.)

Usually `makeindex` works in conjunction with a style file. [In fact, the syntax of FWEB's style file (see Section 12.3 [Style], page 112) was motivated by that of `makeindex`.] When the '`-XI`' option (see Section 4.2.68 [-X_], page 34) is used, FWEAVE will *create* an appropriate style file for `makeindex`. (The default name of '`test.sty`' can be overridden by the style-file parameter `makeindex.sty`.) To run `makeindex` on the index data for '`test.web`' and create the output file '`index.ind`', one would thus say

```
makeindex -s test.sty -o index.ind test[.idx]
```

It's important to use the '`-o`' option with a name different than the original file name, because it simplifies the construction of the skeleton file '`index.tex`' that prints the stand-alone index.

FWEAVE writes '`test.sty`' because the contents of that file may depend on parameter settings in FWEB's style file '`fweb.sty`'. FWEB's style vocabulary includes all parameters understood by `makeindex`. If a `makeindex` parameter is called '`param`', one references it in '`fweb.sty`' by '`makeindex.param`'. Thus, to change the '`headings_flag`' of `makeindex`, one would put into '`fweb.sty`' a line like '`makeindex.headings_flag = 1`'. To see a list of all `makeindex`-related parameters, say '`fweave -Zmakeindex`' (see Section 4.2.70 [-Z_], page 35). Remember, *do all* `makeindex` *customizations in* '`fweb.sty`'*; the actual style file* '`test.sty`' *that will be read by* `makeindex` *is written automatically by* FWEAVE.

The '`.idx`' file will contain a list of entries that begin with '`\indexentry`' (more precisely, the value of the parameter '`makeindex.keyword`'). The general form is

```
\indexentry{sort key:identifier expression|macro}{page number}
```

Typical entries are

```
\indexentry{istream:"\>{istream}|pg{}{}}{1}
\indexentry{main:"\>{main}|pg{}\underline}{1}
\indexentry{pow:"\@{pow}|pg{}{}}{2}
\indexentry{z:"\"|z|pg{}\underline}{2}
```

Here the colon is the value of '`makeindex.actual`'; it separates the sort key (before the colon) from the actual expression to be printed. The macros such as '`\>`' typeset the identifiers in the appropriate way, depending on their use in the code. Note that the backslashes are quoted with the value of '`makeindex.quote`', which is by default the double quote.

Although one might guess that the typesetting macros such as '`\>`' would be defined in '`fwebmac.sty`', that is not true. Rather, for various technical reasons they are equated to macros in '`fwebmac.sty`' as one of the operations of the '`\Wbegin`' macro that is executed at the beginning of every `tex` file output by FWEAVE. For example, '`\Wbegin`' does the equivalent of '`\let\>\Wid`'. Unfortunately, without further action that equating would be forgotten by a LaTeX run made on the output '`index.ind`' of `makeindex`. For that reason, FWEAVE appends the appropriate '`\Wequate`' macro to the end of '`makeindex.preamble`'. This is one specific instance that necessitates that FWEAVE write the `makeindex` style file.

Each of the '`\indexentry`'s contains the encapsulation character '`|`' (the value of '`makeindex.encap`'). By the conventions of `makeindex`, everything between the encapsulation character and the closing right brace defines a macro expression that acts on the page number. E.g., the general form above generates the command '`\macro{`*page number*`}`'. The specific macro construction output by FWEAVE is

`\pg{}{`*possible action macro*`}{`*page number*`}`

Here the name '`pg`' is the value of '`makeindex.page`'. The *action macro* is something like '`\underline`', which would be used by FWEAVE to underline the page number to indicate where a variable is defined. A default definition of '`\pg`' is given is '`fwebmac.sty`'. It is a three-argument macro, '`\def\pg#1#2#3{...}`', where the arguments are as follows:

`#1` — Integer file identification number
`#2` — Action macro.
`#3` — Page number.

The definition should contain the construction '`#2{#3}`'—i.e., the page number must be the argument of the action macro. The first argument is left empty in the '`.idx`' file written by FWEAVE. This can be filled in later by the utility `idxmerge` (see Section 11.3 [Merging indexes], page 105) that merges the indices from several `web` files. For example, in a master index one might ultimately print page numbers like '`II.5`', where '`II`' refers to a file such as '`test2.web`'. To aid this merging process, the root name of the `web` file is written as a comment at the top of the '`.idx`' file output by FWEAVE.

11.3 Using the `idxmerge` utility to merge indexes

In a large project, one may maintain and work with several FWEB files. It may be useful to produce a global index that spans all of those files. To this end, the utility `idxmerge` and associated LaTeX package `idxmerge` are supplied with the FWEB distribution.

11.3.1 Using `idxmerge`: Summary

As quick reference for those who have already plowed through the following details, here is a summary of the procedure. To print a stand-alone index by merging the indexes from several `web` sources, do the following. First, create, if necessary, a file '`index.tex`' that `\inputs` '`index.ind`'. Then:

```
fweave -XI test1.web
fweave -XI test2.web
fweave -XI test3.web

idxmerge -oindex test1.idx test2.idx test3.idx
        % Creates index.ind and index-names.tex.
makeindex -s test1.sty index
latex index
```

Note the use of the '-XI' option. For further background, see the previous section, Section 11.2 [Using makeindex], page 103.

11.3.2 Using idxmerge: Details

Suppose one has three files, 'test1.web', 'test2.web', and 'test3.web'. To use idxmerge, weave each of the files separately, using the '-XI' option to create 'test*.idx' and 'test*.sty'. Then say

```
idxmerge -oindex test1.idx test2.idx test3.idx
```

This creates two output files: 'index.idx', and 'index-names.tex'. idxmerge first sorts the list of file names. It then writes one entry into 'index-names.tex' for each file, of the form

```
\idxname{n}{file_namen}
```

This file can be \input by the \topofindex command (for an example, see the LaTeX2e package idxmerge) (supplied with the FWEB distribution) and used to create a list of the merged files.

Then it merges the \indexentry commands from each of the input files into 'index.idx', filling in the integer file identifier *n* (the position of the file in the sorted list) into the first argument of the \pg macro. One can now say

```
makeindex -s test1.sty index
```

This creates 'index.ind', which can be processed by, for example, a simple modification of the simple LaTeX template given above in Section 11.2 [Using makeindex], page 103. The only difference is that the package idxmerge was used; in that file, the macros \topofindex and \idxname are appropriately defined to print out a numbered list of the merged files to cross-reference into the numerical file- and page-number entries in the body of the index. Here is an example (provided in the FWEB distribution):

```
% index.tex --- skeleton for printing a stand-alone index.
\documentclass{article}
\usepackage{fwebmac,idxmerge}

\begin{document}

\input{\jobname.ind}

\end{document}
```

12 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. See Section 12.2 [Initialization], page 108.

The style file (called 'fweb.sty' by default; see Section 4.2.71 [-z], page 35) is intended to provide more local customization, perhaps differing for each source file and group of source files. The style file does not contain command-line options; rather, it contains parameter settings that override FWEB's defaults. The '-p' option (see Section 4.2.46 [-p], page 28) may be used to specify a style-file entry in '.fweb' (i.e., a global value for all source files) or on the command line (i.e., a value used for a single run).

The order of processing is:

1. Evaluate environment variables. See Section 12.1 [Environment variables], page 107.
2. Read '.fweb' and remember its contents; sort those into three groups: options beginning with '-', beginning with '&', and beginning with a letter (file names) . See Section 12.2 [Initialization], page 108.
3. Process '.fweb' options beginning with '-' (or '+', for backward compatibility), except for '-p'.
4. Read and process command-line options, except for '-p'. See Section 4.2 [Options], page 15.
5. Process remaining '.fweb' options (either file names, or options beginning with '&').
6. Process any '-p' options from '.fweb'. See Section 4.2.46 [-p], page 28.
7. Process the style file. See Section 12.3 [Style], page 112.
8. Process any '-p' options from the command line.

Unfortunately, because not all options are processed immediately when they are read, errors may not show up when one expects. For example, nothing is actually processed while '.fweb' is being read; its contents are just being stored. It could therefore happen that a syntax error in entering a '-p' option in '.fweb' may not be reported until after the style file has been read, possibly confusing the user as to the source of the error.

12.1 Environment variables

FWEB_HDR_INCLUDES — Colon-delimited list of directories for the C preprocessor (in the form of gcc) to search for '#include' header files. This is used in conjunction with the '-H' option; see Section 4.2.17 [-H_], page 20. (One can append to this list by means of the '-I' option, provided that option comes *after* the '-H'; see Section 4.2.19 [-I_], page 21.)

`FWEB_INCLUDES` — Colon-delimited list of directories to search for '`@i`' include files. (One can append to this list by means of the '`-I`' option, provided that option comes *before* any use of '`-H`'; see Section 4.2.19 [-I_], page 21.)

`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 the style file resides. If not defined, the current directory is used.

12.2 Initialization

Although some aspects of FWEB's behavior are hard-coded, many can be changed and/or initialized by the user.

12.2.1 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 '`-`' (or a '`+`' for backward compatibility), it is processed *before* the actual command-line options; if it begins with '`&`' or is a file name, it is processed after. Generally, '`.fweb`' options should begin with '`-`' so that one may override them from the command line. The '`%`' sign begins a comment terminated by the end-of-line.

12.2.2 Memory allocation

The command-line option '`-y`' (see Section 4.2.69 [-y], page 35) 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 one increase a default allocation. To query the present allocations of variable *aa*, just say '`-y`*aa*'. To query everything, say '`-y`'.

This whole scheme is somewhat annoying. In most cases, dynamic arrays should be reallocated automatically. That can be done without too much difficulty, but I was reluctant to try it for Version 1.61 in fear of breaking something. Please wait for the year 2000.

If one uses '`-y`' to examine the maximum permitted values of these parameters, one will note the magic number 10239 appearing occasionally. This number is a bit less than 64K/5; it is a signature of an inherently 32-bit design that goes back to Knuth. Unfortunately, this number can't be increased without some radical redesign. Wait for the year 2100.

12.2.2.1 '`-yb`': Maximum bytes for identifiers, index entries, and module names

Unique identifiers, index entries, and module names are stored contiguously in a large memory area, the size of which is controlled by '`-yb`'. The default may need to be increased for very large source files, or decreased to squeeze things into a personal computer. See also Section 12.2.2.20 [-yn], page 111.

12.2.2.2 '-ybs': Size of the change buffer, in bytes

Information from change files is read into the change buffer, whose size is controlled by '-ybs'. It should not be necessary to change this unless an error message specifically tells one to do so.

12.2.2.3 '-ycb': Size of line buffer for C output, in bytes

FTANGLE outputs lines of a fixed maximum length. It attempts to split them in a reasonable way, dependent on the language. When it absolutely can't figure out how to split the line, it will issue a warning message and split it anyway. The '-ycb' option controls the maximum output line length for C and C++.

The analogous command '-yxb' controls the output line length for TeX and the verbatim mode. See Section 12.2.2.29 [-yxb], page 112.

12.2.2.4 '-ycf': Size of a Ratfor buffer, in bytes

The sizes of buffers used by RATFOR for constructing messages about the commands it is expanding are controlled by '-ycf' and '-ycg'.

12.2.2.5 '-ycg': Size of another Ratfor buffer, in bytes

The sizes of buffers used by RATFOR for constructing messages about the commands it is expanding are controlled by '-ycf' and '-ycg'.

12.2.2.6 '-yd': Increment for expanding the dots table

The "dots" table is used for FORTRAN to hold information relating to "dot" operators such as '.NE.'. In FORTRAN–90, additional such operators can be added by the program, so the table can grow dynamically. The '-yd' option controls how many additional entries are made available each time the table size needs to be reallocated.

12.2.2.7 '-ydt': Maximum number of deferred macro tokens

Deferred FWEB macros are ones defined in the code part rather in the definition part. (Their use is normally prohibited; see Section 4.2.59.1 [-TD], page 31.) '-ydt' controls how many bytes are set aside for the storage of these replacement text of those macros. See also Section 12.2.2.8 [-ydx], page 109.

12.2.2.8 '-ydx': Maximum number of deferred macro texts

'-ydx' controls how many deferred macros are permitted. See also Section 12.2.2.7 [-ydt], page 109.

12.2.2.9 '-yid': Maximum depth of file inclusion

Files included by '@i' can themselves contain '@i' commands, to a nesting level controlled by '-yid'.

12.2.2.10 '-yif': Maximum number of unique include-file names

The number of unique file names appearing in '@i' commands is controlled by '-yif'.

12.2.2.11 '-ykt': Stack size for FTANGLE

FTANGLE uses a stack to deal with the web of module names—i.e., a named section can refer to another module name. The size of this stack is controlled by '-ykt'.

12.2.2.12 '-ykw': Stack size for FWEAVE

FWEAVE's stack handles the possibilities that code mode can be embedded in a module name, or vice versa. The maximum nesting level for such mode changes is controlled by '-ykw'.

12.2.2.13 '-yll': Line length for FWEAVE's output, in bytes

'-yll' controls the length of each line in the .tex file output by FWEAVE.

12.2.2.14 '-yln': Maximum length of module names or strings, in bytes

When each module name or string is parsed, it is stored temporarily in a buffer whose length is controlled by '-yln'.

12.2.2.15 '-ylb': Maximum number of nested loops in RATFOR

In RATFOR, various loops such as 'while' are translated into their FORTRAN equivalents. '-ylb' controls the maximum nesting level of such expandable constructions.

12.2.2.16 '-ylx': Maximum length of expressions that can be expanded with the post-increment operators of FORTRAN or RATFOR

FORTRAN and RATFOR can expand expressions such as 'x(i) += dx' into their FORTRAN counterparts such as 'x(i) = x(i) + dx'. It does so in a very straightforward way, by copying the expression to the left of the equals sign. '-ylx' controls the maximum size of that expression.

12.2.2.17 '-ym': Maximum number of sections

'-ym' limits the maximum number of sections, both named and unnamed. (Each unnamed section is counted separately.) The absolute maximum number of sections is 10239, probably one of the most stringent restrictions in FWEB's design. (This number is a bit less than 1/5 of 64K.)

12.2.2.18 '-yma': Maximum number of arguments to FWEB macros

The maximum number of arguments to FWEB macros (defined by '@m') is limited by '-yma'.

12.2.2.19 '-ymb': Size of the buffer for expanding FWEB macros

The expansion of each FWEB macro is done in a buffer whose size is controlled by '-ymb'. (In some situations, particularly in RATFOR, more than one such buffer can be open at once.)

12.2.2.20 '-yn': Maximum number of identifiers and module names

A structure is associated with each unique identifier and module name. The maximum number of such structures is controlled by '-yn'. See also Section 12.2.2.1 [-yb], page 108.

12.2.2.21 '-ynf': Maximum number of open output files

In addition to FTANGLE's usual output file—e.g., `test.c`—additional files may be opened by means of the '@O' (see Section 5.5.20 [ATO_], page 47) or '@o' (see Section 5.5.21 [ATo], page 47) commands. Depending on the situation, some of these files may remain open simultaneously. The maximum number of such files is controlled by '-ynf'.

12.2.2.22 '-yop': Maximum number of entries in the table for operator overloading.

In FWEAVE, the appearance of an operator can be changed (*overloaded*) by means of the '@v' command (see Section 5.5.27 [ATv], page 49). Each such operator is entered into a table, the maximum size of which is controlled by '-yop'.

12.2.2.23 '-yr': Maximum number of cross-references

The Index cross-reference information (in which sections each identifier is used or defined) is maintained in a large array of structures, one structure for each cross-reference. The maximum number of cross-references is controlled by '-yr'.

12.2.2.24 '-ys': Maximum number of scraps

The maximum number of scraps is controlled by '-ys'. For a discussion of scraps, see Section 4.2.2 [-1], page 16.

12.2.2.25 '-ysb': Size of style-file input-line buffer

The maximum length of each input line of the style file (`fweb.sty` by default) is controlled by '-ysb'.

12.2.2.26 '-ytt': Maximum number of tokens that FTANGLE can process

A *token* is an identifier, numerical constant, operator, etc. FTANGLE must read in and store all tokens in the entire source file, because they can be output in a different order than they are input. The maximum number of tokens is controlled by '-ytt'.

12.2.2.27 '-ytw': Maximum tokens in the current section being processed by FWEAVE.

Unlike FTANGLE, FWEAVE need only read in one section at a time. The maximum number of tokens in any section is controlled by '-ytw'.

12.2.2.28 '-yx': Maximum number of texts

For FTANGLE, a *text* is either the replacement text of a macro, or the contents of a named section. The maximum number of such texts is controlled by '-yx'.

For FWEAVE, a *text* is a phrase that arises from combining primitive scraps during the translation stage of phase 2.

For both processors, the absolute maximum number of texts is 10239.

12.2.2.29 '-yxb': Size of line buffer for TEX and verbatim output

This option is like '-ycb' (see Section 12.2.2.3 [-ycb], page 109), but controls the size of the output line for the TEX ('@Lx') and verbatim ('@Lv') languages.

12.3 The Style file

A *style file* (default name 'fweb.sty') may reside in the user's current directory (or the directory specified by the environment variable FWEB_STYLE_DIR). The default name can be changed by the command-line option '-z' (see Section 4.2.71 [-z], page 35).

The style file is processed after all command-line options have been processed, except that the command-line option '-p' (see Section 4.2.46 [-p], page 28) gets special treatment. Note that that option buffers up style-file entries (i.e., one may use more than one '-p' option). '-p' options placed in '.fweb' are treated as residing in a temporary file that is read just *before* the local style file; thus, those behave as 'global' style-file entries that will be overridden by a matching entry in the local style file. '-p' options on the command line will be processed *after* the local style file, thus override corresponding options in either '.fweb' or the local style file.

To summarize the previous discussion, the local style file is intended to contain settings that are common to a particular source file. Settings common to all source files can be put into '.fweb' by means of the '-p' option. To override a setting for a single run, use a '-p' option on the command line.

Style-file entries have the form

keyword [=] value

The equals sign is always optional. The 'value' is usually a double-quoted string, but may sometimes be an integer or a single-quoted character. For example,

```
LaTeX.class.options = "twoside"
LaTeX.package "indentfirst,multicol"
mark_defined.fcn_name 0
line_char.N 'C'
color.error = "red"
Color.red = "\e[01;31m"
```

The syntax is completely free-form. Periods within keywords are precisely equivalent to underscores, but are useful heuristically for associating a structure-like hierarchy to some of the commands. Non-printable characters in strings can be specified as octal constants (e.g., '`\033`'), hexadecimal constants (e.g., '`\x1B`'), or one of the ANSI escape sequences '`\a`', '`\b`', '`\f`', '`\n`', '`\r`', '`\t`', and '`\v`'. The non-ANSI escape sequence '`\e`' (escape) is also supported; that is particularly useful for color processing (see Section 12.3.7 [Color], page 117).

Various of the style-file parameters take a language subscript. Those are

`C` C

`Cpp` C++

`N` FORTRAN-77

`N90` FORTRAN-90

`R` RATFOR-77

`R90` RATFOR-90

`V` Verbatim

`X` TeX

Thus, `line_char.N` is the comment character for FTANGLE's `line` commands (see Section 12.3.8.4 [line_char], page 119), for FORTRAN-77 code.

Unfortunately, the descriptions of the parameters aren't all completed yet. To query the default values, say '`ftangle -Z`' (see Section 4.2.70 [-Z_], page 35).

12.3.1 Customizing FWEAVE's index

In the following, '`???`' denotes the name of various subparameters.

12.3.1.1 `index.`*???*

`index.name` is the name of the index section. This string is used in `\Wbegin` to initialize the TeX macro `\INDEX`. The index section is recognized by matching, for a starred section, the actual section name against the contents of `\INDEX`. When they match, a new page and two-column mode are begun. These rules imply that the last section of one's source file can be titled '`\INDEX`', as in

```
@* \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, as for example '`#.ndx`'.

`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.

12.3.1.2 `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.

12.3.1.3 `group_skip`

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

12.3.1.4 `item_0`

`item_0` is the TEX command to begin an index entry.

12.3.1.5 `language.`*???*

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

12.3.1.6 `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.

12.3.1.7 `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.

12.3.2 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.

12.3.3 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.

12.3.4 Customizing cross-reference subscripts

When FWEAVE pretty-prints code, it can attach cross-reference subscripts to various kinds of identifiers such as function or macro names. [A bullet (•) for a subscript indicates that the name was defined in the current section.] The actual marking of the cross reference is done by the command '`@[`' (see Section 5.7 [AT[], page 51). This is usually done implicitly; for example, the commands '`@a`', '`@d`', and '`@m`' issue an implicit '`@[`'. (See the discussion of '`@a`' in Section 5.4.4 [ATa], page 40.) In C, various declarations of variables also result in such an implicit mark.

Various nuances in the type (possibly underlined) used for the subscript give a hint about what kind of identifier FWEAVE thinks it's working with. For more information about the typesetting conventions, see the definition of the primitive macro '`\W@IN`' in '`fwebmac.web`'.] The following flags select which identifiers are so subscripted.

To see the default values of these parameters, say '`ftangle -Zmark_defined`'. To turn off the subscripting operations completely, use the '`-f`' option (see Section 4.2.16 [-f], page 20).

(Discussion to be completed.)

12.3.5 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. (Please try not to actually edit '`fwebmac.sty`' itself; it is produced automatically from '`fwebmac.web`'. And please don't edit that file either!)

12.3.5.1 `format.`*???*

The `format` parameters are strings that specify the macro to be used to pretty-print various kinds of identifiers. These macro names are usually written automatically by FWEAVE, but they may also be used directly by the user in the TEX documentation. One can see their default values by typing '`ftangle -Zformat.`'. For example, the default value for `format.typewriter` is `"\\."`.

The macro names defined by the `format` fields are *not* defined in '`fwebmac.sty`'. They are *dummy names*, and can be changed to any other name not already in use without affecting the operation of FWEB. This ability is necessary because other packages might usurp macros like `\.` for their own purposes.

Thus, FWEAVE normally writes out the macro `\.` to typeset a string. Suppose, however, that some user package uses `\.` for something else. (One might realize this when LATEX crashes when it encounteres a `\.` that was written automatically by FWEAVE.) To fix this problem, put into '`fweb.sty`' the lines

```
format_KEYWORD = "\\WTT"
format_keyword = "\\WTT"
format_typewriter = "\\WTT"
```

Here `\WTT` can be any name not already in use; *you need not (and should not)* give a definition for `\WTT`.

Macros like `\.` or `\WTT` are given their values during the execution of the `\Wbegin` macro that begins the output from FWEAVE. The style-file values are written as arguments to that macro, and essentially a command like `\let\.\Wtypewriter` is executed, where the internal macro `\Wtypewriter` is defined in '`fwebmac.sty`'. If you want to change the way FWEB typesets a particular kind of identifier, you must redefine the *internal* macro name, not the one used in the `format` parameters.

Here are the internal macros used by '`fwebmac.sty`' to typeset the various kinds of identifiers. The associated style-file parameters are shown in parentheses.

`\Wid` ordinary identifiers (`format.id`)

`\WID` completely upper-case ordinary identifiers (`format.ID`)

`\Wshort` single-character ordinary identifiers (`format.short_id`)

`\WidD` outer macros (`format.outer_macro`)

`\WIDD` completely upper-case outer macros (`format.outer_macro`)

`\WidM` FWEB macros (`format.WEB_macro`)

`\WIDM` completely upper-case FWEB macros (`format.WEB_macro`)

`\Wreserved`
reserved words (`format.reserved`)

`\WRESERVED`
completely upper-case reserved words (`format.RESERVED`)

`\Wintrinsic`
library/intrinsic function names (`format.intrinsic`)

`\Wkeyword`
certain Fortran keywords (`format.keyword`)

`\WKEYWORD`
completely upper-case keywords (`format.KEYWORD`)

`\Wtypewriter`
character strings (`format.typewriter`)

12.3.5.2 `indent.`*???*

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

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

12.3.5.3 `LaTeX.`*???*

For LaTEX2e, the default document class can be overridden by `LaTeX.class`. The default class is `article`, and FWEB has not been tested with other document classes, except minimally with `revtex` (see Section 10.1.3.2 [REVTeX], page 94).

Options to the document class can be specified by `LaTeX.class.options`.

User packages can be given by `LaTeX.package`.

Options to user packages can be specified by `LaTeX.package.options`. There may be just one `LaTeX.package` command and just one `LaTeX.package.options` command. If it is necessary to issue multiple such commands, then put them into `doc.preamble`. See the discussion in Section 10.1.3.1 [Document class], page 93.

When running under LaTEX prior to LaTEX2e (or with REVTeX; see Section 10.1.3.2 [REVTeX], page 94), the document is (effectively) begun by the command `\documentstyle[options]{style}`. The options field can be specified by `LaTeX.options`; the style field by `LaTeX.style`.

12.3.6 Remapping control codes

Control-code remappings are sophisticated and unwise. They are mostly intended for the developer, so are not explained here.

12.3.7 Color output

In the design of FWEB, provision has been made for writing various messages to the terminal in color—e.g., serious error messages might appear in red. This feature was motivated by the color `ls` of Linux. It is installed automatically if the `termcap` library is present.

Messages output from FWEB are ranked according to an internal message-type table; each type can be associated with a color that can be changed in the style file. Presently, the message types (hopefully self-explanatory) are

```
ordinary
program_name
mod_name
info
warning
error
fatal
mod_num
line_num
in_file
include_file
out_file
timing
```

The associated style-file parameters are the above names prefaced by '`color.`'—e.g., `color.warning`. Each of those has a default value, such as `color.error = "red"`. Those defaults can be displayed by saying '`ftangle -Zcolor`'.

What the color actually means in practice depends on the *color mode*, set by the '`-C`' option (see Section 4.2.8 [-C_], page 18). That selects one of several primitive palettes, as follows:

0
: **No color**; ordinary black-and-white output. This is the default (and the mode used when the `termcap` library is not present).

1
: **ANSI color**. With a color terminal that supports ANSI color escape sequences, one has available the following colors: `"black"`, `"red"`, `"green"`, `"yellow"`, `"blue"`, `"magenta"`, `"cyan"`, `"white"`, and `"default"`. These are displayed with bold attribute (that is, bright, not dim). '`"default"`' stands for the usual black on white background, or vice versa.

2
: **Bilevel**. This is for terminals that don't support true color, but do support a double-bright mode and reverse video. Colors are mapped onto various combinations of those two display attributes, according to an internally defined scheme. For example, `"red"` is mapped onto the pair of escape sequences '`md`', '`mr`' (double-bright mode in reverse video).

3
: **Trilevel**. As above, but adds underlining capability.

4
: **User-defined colors**. This implements a minimal set of defaults. It is intended that the user add definitions in the style file to override those defaults.

The mechanism is intended to work with systems that support the `termcap` library. The terminal is controlled by writing appropriate escape sequences to it. The style-file parameters that store the escape sequences are the color name preceded by '`Color.`' (note the upper case '`C`')—e.g., '`Color.red`'. For cases like reverse video (standard termcap abbreviation '`mr`'), the escape sequences are determined by querying the termcap database (usually '`/etc/termcap`') through the `termcap` library functions. For ANSI color (color mode = 1), ANSI escape sequences are hard-coded into FWEB. One can see the escape sequences FWEB assigns to colors by saying '`ftangle -ZColor`'.

For any non-zero color mode, one can override FWEB's default choices for color mappings and escape sequences by redefining one or more of the `Color` parameters in the style file. The escape sequences can either be specified in raw form—e.g., for color mode = 1, a default is `Color.red = "\e[01;31m"`—or in the form of a sequence of two-character abbreviations that are defined in the termcap documentation—e.g., for modes 2 and 3, the default is `Color.red = "mdmr"`. (When one displays that with the '`-Z`' option, FWEB will display the actual escape sequences that it determines from the termcap database, not the abbreviations. For both input and output, note that one may use the non-ANSI escape sequence '`\e`' to represent the escape character '`\033`'.)

When one says '`-ZColor`', for color modes 1–3 all of the parameters are listed as modified, even if the user redefines none. That occurs because the defaults are overwritten internally when the color mode is set.

FWEB's configuration script attempts to determine whether the termcap library is present; if not, they link in dummy termcap routines ('`termcap0.web`'). To override this behavior, change the appropriate lines in '`defaults.mk`', produced by the command `./configure`.

Color message output is not fully debugged (it's a frill, after all), so some messages that should reasonably be colored may not be so in the present release.

12.3.8 Miscellaneous style-file parameters

There are a variety of miscellaneous parameters.

12.3.8.1 `ASCII_Fcn`

See Section 5.6.2 [ATdquote], page 51.

12.3.8.2 `cchar`

Continuation character for FORTRAN code output.

12.3.8.3 `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.

12.3.8.4 `line_char.`*l* (FTANGLE)

By default, FTANGLE outputs comments indicating line numbers in the `web` file from which the tangled output comes. (This information can be used by debuggers, especially those for C and C++, to correlate error messages to the `web` source.) The `line_char` parameter sets the comment character that begins the line comment.

12.3.8.5 `line_length.`*l* (FTANGLE)

This parameter is used by the FORTRAN-like languages to control the length of the output line in the '`.f`' file produced by FTANGLE. For FORTRAN-77, its default value is the venerable 72. For FORTRAN-90, its default is 73. Using that value makes it possible to generate code that is compatible with both fixed- and free-form format (by continuing lines with an trailing ampersand in column 73 and another ampersand in column 6 of the next line).

12.3.8.6 `meta.`*???.?*, `meta.`*???*`.hdr.`*?* (FTANGLE)

These parameters customize the treatment of meta-comments. Fundamentally, meta-comments consist of material enclosed by '`@(...@)`'. The header information usually written at the top of the file output by FTANGLE (see Section 4.2.59.4 [-Tv], page 31) is also treated as a meta-comment. For that header material, a separate set of parameters is provided, such as `meta.top.hdr`.

`meta.top.`*l* specifies text that precedes material enclosed by '`@(...@)`'. Here *l* is one of the standard language subscripts (see Section 12.3 [Style], page 112) such as `N90`.

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

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

12.3.8.7 `outer.`*???*

FTANGLE converts '`@d`' (see Section 5.5.6 [ATd], page 42) to `outer.def`, and '`@u`' (see Section 5.5.26 [ATu], page 49) to `outer.undef`.

12.3.8.8 `protect.`*?*

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

12.3.8.9 `suffix.`*?*

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

12.3.8.10 `macros`

The default name of the macro package to be read in. [This is usually '`fwebmac.sty`' (see Section 10.1.2 [fwebmac.sty], page 92), but can be overridden by the command-line option '`-w`'; see Section 4.2.66 [-w], page 34.]

12.3.8.11 `limbo.begin`, `limbo.end`

'`limbo.begin`' is TEX material to be printed at the beginning of the limbo section, just before the text from '`@l`' commands. See Section 5.5.14 [ATl], page 45. (This command was previous called just '`limbo`', and that still works.)

Similarly, '`limbo.end`' is printed at the end of the limbo section.

Thus, the beginning of the file output by FWEAVE looks like this:

```
\input fwebmac.sty

\Wbegin{...}
   [contains \documentclass, \usepackage, <doc.preamble>, \begin{document}]

<limbo.begin>
[contents of any @l commands]
[user's TeX commands from the limbo section]
<limbo.end>
```

The '`limbo.end`' command is useful for printing the entire document in two-column format. For more discussion, see Section 10.1.3.5 [LIndex], page 96.

12.3.8.12 `meta.`*???* (FWEAVE)

(To be finished.)

12.3.8.13 `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`.

12.3.8.14 `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*.

12.3.8.15 `null_file`

The name of the null file or device. For more discussion, see Section 3.3 [Change files], page 13.

12.3.9 Automatic file name completion

For more information, see Section 4.2.14 [-e], page 19.

13 USAGE TIPS and SUGGESTIONS

In this section are collected various tips and suggestions to help one make full use of FWEB. Additional hints broken down by each supported source language can be found in Chapter 8 [Languages], page 83.

13.1 Converting an existing code to FWEB

To convert an existing code to FWEB, one should do the following. (The following simple procedure assumes that one puts 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`' or '`@c++`'.
3. Place an '`@a`' command (switch into unnamed code) 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. (One 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 ('`@e`'), or pseudo-colons ('`@:`') (see Section 10.2.1 [Pseudo-operators], page 101).
9. Consider using FWEB's built-in macro preprocessor (see Chapter 7 [Macros], page 62) to make your code more readable—for example, replace raw numerical constants by symbolic names.
10. Scientific programmers may benefit from built-in macro-like functions like `$PI`; see Section 7.2.3 [Built-in functions], page 66.
11. 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. The '`-nC`' option (see Section 4.2.36 [-nC], page 25) may be helpful.

13.2 Programming tips and other suggestions

1. Learn how to use the GNU `info` browser to access the on-line documentation.
2. Read the list of new features and changes for the last several releases. See Chapter 14 [New features], page 124.
3. Periodically check `ftp.pppl.gov:/pub/fweb/READ_ME` for bug reports and other news. Make bug reports! See Chapter 15 [Support], page 131.
4. If you have a color terminal, try the option '`-C1`' (see Section 4.2.8 [-C_], page 18, see Section 12.3.7 [Color], page 117).
5. Any option in '`.fweb`' that is intended to be processed *after* the command-line options should begin with '`&`' rather than '`-`'. (This is rarely necessary.) See Section 12.2 [Initialization], page 108
6. Put standard command-line options into '`.fweb`'. Also put there standard style parameters—e.g.,

   ```
   -pindex.tex "#.ndx"
   -pmodules.tex "#.mds"
   -pcontents.tex "#.cts"
   ```
7. Learn how to use the style file. See Section 12.3 [Style], page 112.
8. 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 4.2.82 [Info options], page 37.
9. Begin all FWEB sources with invisible commentary bracketed by '`@z...@x`'. See Section 5.5.31 [ATz], page 50.
10. Always include an explicit language-setting command in the limbo section. Under normal circumstances, do not set the language from the command line. See Chapter 8 [Languages], page 83.
11. 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. If a block of code is longer than that, split it up using named modules.
12. It's easy to define macros from the command line to expedite conditional preprocessing. See Section 4.2.27 [-m], page 23.
13. Use the preprocessor construction '`@#if 0...@#endif`' to comment out unwanted code. See Section 7.3 [Preprocessing], page 80.
14. For logical operations with the preprocessor, use '`||`', not '`|`'.
15. It's conventional to identify the ends of long preprocessor constructions as follows:

   ```
   @#if A
   ```

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

16. To debug an errant FWEB macro, use the built-in function '`$DUMPDEF`'. See Section 7.2.3.14 [$DUMPDEF], page 70.
17. Use '`@?`' for compiler directives. See Section 5.8.4 [AT?], page 53. Use the style-file parameters '`cdir_start`' to specify information that will be written out at the beginning of the line. See Section 12.3.8.3 [cdir_start], page 118.
18. 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 60.
19. The meta-comment feature '`@(...@)`' provides a poor-person's alignment feature. But it doesn't work very well, and it's not in the spirit of TEX; learn to use '`\halign`' or the LaTEX alternatives.
20. In FORTRAN, use '`#:0`' to declare readable alphabetic statement labels. See Section 7.2.2 [Tokens], page 65 and Section 4.2.75 [-colon], page 36.
21. When mixing languages, define the language of a module at the highest possible level—e.g., in the unamed module, not after '`@<...@>=`'.
22. Use LaTEX. Plain TEX is no longer supported. Upgrade to LaTEX2e. See Section 10.1.3 [LaTeX], page 93.
23. If you are reading this documentation from printed pages, make sure it's also installed as an Info package on your system so it can be read interactively with `emacs`. You can also read it through a World-Wide Web browser such as Netscape. For the address, see Chapter 15 [Support], page 131.

13.3 Features for scientific programming

FWEB contains a few features particularly intended for scientific programming.

1. Several built-in functions generate numerical constants. See '`$PI`' (Section 7.2.3.47 [$PI], page 76) and '`$E`' (Section 7.2.3.15 [$E], page 70).
2. Several built-in functions perform mathematical manipulations. See '`$EXP`' (Section 7.2.3.18 [$EXP], page 71), '`$POW`' (Section 7.2.3.48 [$POW], page 76), '`$SQRT`' (Section 7.2.3.56 [$SQRT], page 77), '`$LOG`' (Section 7.2.3.37 [$LOG], page 75), '`$LOG10`' (Section 7.2.3.38 [$LOG10], page 75), '`$MAX`' (Section 7.2.3.40 [$MAX], page 75), and '`$MIN`' (Section 7.2.3.41 [$MIN], page 75).
3. The do-loop macro '`$DO`' may be useful. See Section 7.2.3.13 [$DO], page 69.
4. C-style array indices can be used by means of the '`-n)`' option. See Section 4.2.42 [-n)], page 27.
5. An active bracket feature helps improve the appearance of woven code that uses subscripts and/or superscripts heavily. See Section 4.2.65.3 [-W[], page 33.

14 NEW FEATURES

This info documentation is now accessible on the World-Wide Web; see Chapter 15 [Support], page 131.

Some things that have been added or changed in recent releases are described in the following.

14.1 Version 1.61

14.1.1 Updates to documentation (v1.61)

1. FWEB supports color modes in which messages to the terminal can appear in colors chosen by the user; see Section 12.3.7 [Color], page 117. The color mode is set by the new command-line option '`-C`' (see Section 4.2.8 [-C_], page 18).
2. A previously undocumented feature is that for the C-like and Fortran-like languages, FTANGLE expands the binary notation '`0b...`' to an unsigned decimal number. See Section 2.4 [Phases], page 10.

14.1.2 Redefined commands (v1.61)

A few obscure commands have been slightly redefined. Sorry about that, but it makes for more symmetry and ease of recall, and/or solves some technical problems.

1. Although it was never documented, previous versions permitted either lower or upper case for the '`@`' commands that set the language—e.g., both '`@c`' and '`@C`' worked. Now only the lower-case forms work. (The upper-case forms may have other meanings.)
2. The style-file parameter '`Ext_delimiter`' now begins with an upper-case '`E`'; formerly it was lower-case.
3. The behavior of the optional argument of the `\Title` macro has been slightly redefined. The new, more symmetrical form is

   ```
   \Title[Short title]{Long title}
   ```

   where `Long title` is printed on the title page and `Short title` is used for the running header within the document. See Section 10.1.3.6 [Table of Contents], page 96.
4. The line-break commands '`@/`' and '`@\`' (formerly identical) now behave slightly differently. '`@/`' breaks the line just as it would if the line had been too long and been spontaneously broken. See Section 5.12.2 [AT/], page 55. '`@\`' backspaces one unit of indentation after breaking the line. See Section 5.12.3 [ATbs], page 56. Usually, one should use '`@/`' (sorry; I was previously recommending '`@\`'. For an example in which it is natural to use '`@\`', see Section 5.12.3 [ATbs], page 56.
5. The names of some of the code-typesetting macros in `fwebmac.sty` have been changed to conform to the convention that they should all start with '`W`'. This change will be invisible to you unless you happen to have user macros of your own that start that way or (perish the thought) you have redefined low-level and obscure code in '`fwebmac.sty`'.

14.1.3 New features (v1.61)

This release adds some features for managing large projects, including (i) the `idxmerge` utility that merges indexes produced by several FWEB files, (ii) a mechanism for accessing RCS-like information in the ignorable commentary at the beginning of the file, and (iii) the ability to include FWEAVE-formatted code into a standard LaTeX document. It also fixes a variety of miscellaneous bugs.

1. A stand-alone index file suitable for processing by `makeindex` can be produced by the '`-XI`' option. See Section 11.2 [Using makeindex], page 103.
2. Stand-alone indexes produced by '`-XI`' can be merged with the `idxmerge` utility. See Section 11.3 [Merging indexes], page 105.
3. FWEAVE-formatted code can be included in a standard LaTeX2e document by means of the `fwebinsert` package. See Section 10.1.6.1 [Inserting woven code], page 99.
4. Revision-control-system (RCS) information that appears in the ignorable commentary between the optional '`@z`' and '`@x`' that begin an FWEB file (see Section 5.5.31 [ATz], page 50) is accessible in the body of the file through the built-in function `$KEYWORD` (see Section 7.2.3.30 [$KEYWORD], page 72) and the new commands '`@K`' (see Section 5.5.11 [ATK_], page 44) and '`@k`' (see Section 5.5.12 [ATk], page 44). These features can access RCS-like keywords that are not known to RCS itself, as long as they fit the proper syntax (see Section 5.5.31 [ATz], page 50).
5. The '`-h`' option now permits easy access to the GNU `info` browser if it is installed. See Section 4.2.18 [-h], page 21.
6. Underscored versions of built-in functions have been removed!!! E.g., use `$IF`, not `_IF`. This change was warned about in the last release.
7. Single-character identifiers can now be completely cross-referenced via the '`-W1`' option. See Section 4.2.65.2 [-W1], page 33.
8. Some module warning messages can be eliminated with the '`-W@`' option. See Section 4.2.65.1 [-WAT], page 32.
9. The '`@q`' command (still experimental) has been added to locally turn on or off the the line and module comments in the tangled output. See Section 5.5.22 [ATq], page 48.
10. The level of verbosity of FWEB's informational messages can be controlled with the '`-M`' option. See Section 4.2.26 [-M_], page 23.
11. C/C++ programmers may find the command '`@{`' useful. See Section 5.9 [ATlb], page 53.
12. The '`-nC`' option has been added for FORTRAN users; it kills commented lines at a very early stage in the processing. This can be useful when converting existing codes to FWEB. See Section 4.2.36 [-nC], page 25
13. FORTRAN-90 (see Section 4.2.31 [-n9], page 24) now defaults to free-form syntax.
14. As of the non-beta Version 1.61, free-form FORTRAN-90 now inserts semicolons automatically in the code part. Thus, textbook FORTRAN-90 examples will weave correctly without the annoyance of explicitly terminating

each statement with a semicolon. (If you prefer to put in the semicolons explicitly, use '`--n;`' to turn off the auto-insertion.) See Section 4.2.33 [-n;], page 24

15. The default meaning of the '`-k`' option was changed; now both lower- and upper-case forms of FORTRAN I/O keywords are recognized. See Section 4.2.23 [-k], page 22.
16. Various changes were made to internal code in '`fwebmac.sty`'. This should not affect anyone *unless* you have redefined `fwebmac` macros. If so, you'll have to compare your versions with the present ones. For example, colons as argument delimiters in `\def`s have been removed.
17. It is now (barely) possible to use `\documentstyle{revtex}` instead of the default `\documentclass{article}`. See Section 10.1.3.2 [REVTeX], page 94.

14.1.4 Significant bugs (v1.61)

1. Perhaps the most significant bug is that some high-order (>= 128) characters in strings may not typeset or be processed correctly. This may be an issue for some users of foreign-language packages. The difficulty arises from a design decision made by a previous author. This has at least partly been fixed, but I eschewed a substantial overhaul for fear of breaking other things.

14.2 Version 1.53

This release fixes a relatively small number of obscure bugs in `fweb-1.52-beta`. A few minor enhancements were also made. They include

1. Sections can be numbered by consecutive integers rather than LaTeX's default Dewey-decimal form by saying

   ```
   LaTeX.package = "fwebnum"
   ```

   See Section 10.1.3.4 [Sections], page 95.
2. The '`-H`' option (experimental and incomplete) was added. For C and C++, this option tells FWEAVE to scan `#include` files for '`typedef`' and/or '`class`' definitions. See Section 4.2.17 [-H_], page 20.
3. The '`-k`' option was added. This tells FORTRAN and RATFOR to understand the lower-case forms of I/O keywords such as '`iostat`' (with the exception of '`read`', '`write`', and '`end`'). See Section 4.2.23 [-k], page 22.
4. The '`-n:`' option was added. This tells FORTRAN to place statement labels on a separate line, which is useful when the labels are relatively long. (By default, FORTRAN labels are placed on the same line as the thing they are labeling, which looks good for short labels.) See Section 4.2.34 [-ncolon], page 25
5. The preprocessor command '`@#line`' was added. For C code, this adds an explicit '`#line`' command to the tangled output file. This helps to keep the line numbers between debugger and source file in sync when an FWEB preprocessor statement expands to several lines. See Section 7.2.4 [Debugging with macros], page 79.

An implicit '`@#line`' command is added after each '`@%`' (see Section 5.8.3 [AT%], page 52) that begins a line (this keeps line numbering correct). To override this, use the option '`-T#`'. See Section 4.2.59.6 [-T#], page 31.

6. '`-p`' (style-file) options (see Section 4.2.46 [-p], page 28) on the command line are now processed *after* the local style file. See Section 12.3 [Style], page 112.
7. The functionality of the '`-D`' command was enhanced to include optional arguments that limit the information that will be listed. See Section 4.2.11 [-D_], page 18.

14.3 Version 1.52

This release was issued only as a beta version. It consists mostly of bug fixes. However, there are a few other interesting points.

1. `fwebmac.sty` was enhanced to warn the user to run LaTeX again when the section numbering hasn't yet been brought up to date. I'm not sure I've covered all the bases, but before it didn't complain at all.
2. C++ classes are now formatted (identified as reserved words) on the first pass, so forward references such as

   ```
   @ The class |C|...
   @a
   class C
     {}
   ```

   will now work. Note that **typedef** has done this for a while, although there are still a few glitches.
3. For two years, the documentation has described two control codes as follows:

   ```
   @~ --- inhibit line break.
   @+ --- force an index entry.
   ```

   Apparently the code had these definitions inverted; it has now been brought up to date with the documentation. Fortunately these commands are evidently not heavily used, since no one complained.
4. `fwebmac.sty` was further reworked to interact properly with the user package `multicol`. If in `fweb.sty` one says '`LaTeX.package "multicol"`', then the two-column index is done with `multicol`; this gives various improvements over the `\twocolumn` format that was used previously. Furthermore, it's possible to use '`multicol`' to do one's entire document in two-column format. This turned out to be relatively simple, but one needs to get the commands in the proper order. See Section 10.1.3.5 [LIndex], page 96 for more details. Two-column format substantially cuts down the white space; I saved about 50% on a 200-page code.

   One known glitch with FWEB/`multicol` is that if one selects page-number cross-references instead of LaTeX section numbers, page references such as 98c don't get the 'c' correct. This is presumably not a big deal. At this point, assume that the use of `multicol` is highly experimental.
5. Further bugs in the C and C++ production rules were fixed.

14.4 Version 1.50

1. The syntax for entries in the initialization file '`.fweb`' (see Section 12.2 [Initialization], page 108) has been modified (in a way that is as backward-compatible as possible). Previously, '`+`' meant process the option before the command-line options, '`-`' meant process it after. This convention was somewhat hard to remember, given the statement that any command-line option could be put into '`.fweb`'; furthermore, just about everything in '`.fweb`' should, in fact, be processed before the command-line options. So now both '`+`' and '`-`' mean the same thing, namely process before (and the '`+`' notation should fade away as time goes on). If you explicitly want something to be processed after all command-line options for some tricky reason, begin it with '`&`'. I.e., scan your '`.fweb`' file for any line beginning with '`-`' and replace that with '`&`'.
2. The LaTeX processor ('`-PL`') is now the default.
3. The experimental '`fwebmacL.sty`' macro package supplied with version 1.40 has been substantially reworked and is now the default '`fwebmac.sty`'. Remove any reference to '`fwebmacL.sty`' from your '`.fweb`' file.
4. Support for LaTeX2e is now provided. See Section 10.1.3 [LaTeX], page 93.
5. The style-file parameter `index.name` was added. This is the section name to be given to the Index (see Chapter 11 [Index], page 103), which should be the last major (starred) section. It becomes the contents of the macro `\INDEX`. Therefore, one can end one's source file by saying

   ```
   @* \INDEX.
   ```

6. The '`$IF...`' class of built-in functions was reworked. They should now be more robust, recursive, and intuitive. Simple uses of these functions should work as before. However, complicated uses that depended on tricky things about the order of expansion of arguments may require revision. Carefully compare the descriptions of these functions in the documentation (e.g., see Section 7.2.3.23 [$IF], page 71) with your usage of them in any pre-existing code.

   In some cases, if a previous constructions using `$IF` no longer works, it might work if you say

   ```
   @m $IF(a,b,c) $$IF(a,b,c)
   ```

   and then use `$$IF` in your code. (This forces an extra level of macro expansion.) The same remark goes for `$DEFINE`.

   The old forms '`_IF`' etc. no longer work; convert to '`$IF`'.
7. The option '`-j`' was added. This inhibits multiple inclusions via '`@i`' of the same include file. See Section 4.2.22 [-j], page 22.
8. One now has the ability to change the comment character that begins FTANGLE's '`line`' command. In the style file, say, e.g.,

   ```
   line_char.N '#'
   ```

   to change the default '`*line`' output by FTANGLE in FORTRAN mode to '`#line`'. This could be useful if one runs the C preprocessor on the tangled FORTRAN output.
9. FWEAVE's processing of **typedef** statements in C and C++ was improved.

10. FWEB should now be able to process C++ templates and exception handling, at least in simple situations. The typesetting of C++ references (e.g., '`int&`') was also improved. Please report any difficulties.
11. There were various miscellaneous obscure bug fixes.

14.5 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, *do not use* the underscore forms; they will 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 88
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 5.5.17 [ATN_], page 46.
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 4.2.15 [-F_], page 19.
7. *Converting output tokens to lower case.* See Section 4.2.61 [-U_], page 32.
8. *The built-in functions* '`$E`' *and* '`$PI`'. See Section 7.2.3.15 [$E], page 70, Section 7.2.3.47 [$PI], page 76.
9. *The built-in functions* '`$EXP`', '`$LOG`', *and* '`$LOG10`'. See Section 7.2.3.18 [$EXP], page 71, Section 7.2.3.37 [$LOG], page 75, and Section 7.2.3.38 [$LOG10], page 75.

10. *'`$MAX`' and '`$MIN`' generalized to take arbitrary list of arguments.* See Section 7.2.3.40 [$MAX], page 75, Section 7.2.3.41 [$MIN], page 75.
11. *The marriage-saver option.* In response to a serious user request, see Section 4.2.6 [-B_], page 17.

15 SUPPORT

FWEB is supported by John Krommes, `krommes@princeton.edu`. This project is a definitively *spare-time activity*!!! Bug reports submitted with very short test files will be verified, although not necessarily in real time. For very simple fixes, a change file may be provided. Generally, however, bugs are not fixed until the next release. Releases occur intermittently, depending on my many 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. The next major release of FWEB, Version 2.00, is planned for approximately the year 2000. (You may be relieved to know that, to the best of my knowledge, FWEB does not suffer from the Y2K bug.)

This info documentation is now accessible on the World-Wide Web from

`http://w3.pppl.gov/~krommes/fweb_toc.html`.

You can subscribe to one or both of two FWEB mailing lists, `fweb-users` and `fweb-installers`. To subscribe, send e-mail to `majordomo@pppl.gov`. In the *body* of the message, say, e.g.,

```
subscribe fweb-users
```

You will receive introductory information describing how these lists are intended to be used. To unsubscribe at any time, substitute `unsubscribe` for `subscribe` in the above instructions.

Archive files containing the messages sent to the FWEB mailing lists are kept in

`ftp.pppl.gov:/pub/fweb/archive/fweb-{users,installers}.archive`.

In addition to anonymous `ftp`, these files may be obtained by sending a message to `majordomo@pppl.gov` of the form

```
get fweb-users fweb-users.archive.
```

Appendix A Installing FWEB

Here is the bare-bones installation procedure for UNIX users:

1. Download the `zgip`-compressed `tar` file from `ftp.pppl.gov:/pub/fweb`. The name of the file contains the version number—e.g., '`fweb-1.61.tar.gz`'.

   ```
   ftp ftp.pppl.gov
   bin
   get fweb-1.61.tar.gz
   quit
   ```

2. Uncompress and unpack the tar file:

   ```
   gunzip fweb-1.61.tar.gz
   tar -xf fweb-1.61.tar
   ```

   If the GNU `tar` is installed, these two steps can be combined into

   ```
   gtar -xzf fweb-1.61.tar.gz
   ```

   Unpacking creates the directory '`fweb-1.61`', with at least the two subdirectories '`Web`' and '`Manual`'.

3. Change to the new '`Web`' subdirectory and run the configuration script.

   ```
   cd fweb-1.61/Web
   ./configure
   ```

   '`./configure`' is an `sh` script. It attempts to figure out various local system features automatically, then generates the three files '`defaults.mk`', '`config.h`', and '`custom.h`'; those are used in the `make`. For further information about the operation of '`./configure`', see '`fweb-1.61/READ_ME.FWEB`'.

4. Make and install the release:

   make *[*CFLAGS='*special compiler flags*'*]*
   make install

   If `gcc` is available, it will be used in the `make`; in that case, the default `CFLAGS` should be sufficient. If another compiler is used, ensure that it is run in ANSI-compatible mode, not the old-style Kernighan and Ritchie.

   FWEB compiles on my system without any warnings with '`gcc -ansi -pedantic`'. Please report any compiler warnings from an allegedly ANSI-C environment.

Concept index

R

S

T

U

V

W

Option and command index

Parameter index

N

O

P

S

U

Short Contents

Table of Contents