www.ross.net

F W

Tutorial

Developer

Reference
1 Introduction
2 Interface
3 Scanner
4 Parser
5 Analyser
6 Tangle
7 Weave
8 Shell
9 Commands
10 Glossary
11 References

SEARCH

FunnelWeb Reference Manual

Version 3.2d (9 Jan 2000) for FunnelWeb V3.2

THIS REFERENCE MANUAL provides a concise and precise definition of the functionality of the FunnelWeb literate programming preprocessor. You should refer to this manual when you have specific technical questions about FunnelWeb. The SEARCH facility in the margin provides a quick way to find what you need quickly. If you are not already familiar with FunnelWeb, you may wish to refer to the FunnelWeb Tutorial Manual which provides a structured introduction to FunnelWeb, along with lots of examples and application-specific information. If you want to compile FunnelWeb, refer to the FunnelWeb Developer Manual

FunnelWeb Reference Manual

1 Introduction

FunnelWeb Reference Manual

1.1 Notation

A particular variant of EBNF (Extended Bachus Naur Form) will be used to describe the FunnelWeb syntax. In this variant, literal strings are delimited by double quotes (e.g. "string"), optional constructs by square brackets (e.g. [optional]), and constructs repeated zero or more times by braces (e.g. {zeroormore}). Constructs to be repeated a fixed number of times are enclosed in braces followed by a decimal number indicating the number of times to be repeated (e.g. {sixtimes}6). Constructs to be repeated one or more times are enclosed in braces and followed by a + (e.g. {oneormore}+). The traditional BNF "`::=`" is replaced by the visually simpler "`=`". The traditional BNF angle brackets are abandoned.

Although FunnelWeb allows the special character to be changed using the construct "<special>=", use of "<special>" to refer to FunnelWeb's special character is cumbersome and abstract. To simplify the presentation, the default special character "`@`" is used throughout this chapter to represent the special character.

FunnelWeb Reference Manual

1.2 Terminology

A specific terminology has arisen for dealing with FunnelWeb. Some particularly useful examples are:

> **Journal file:** An output file containing a copy of the output sent to the user's console during an invocation of FunnelWeb. In other systems, this file is sometimes called a "log file".
>
> **Product file:** An output file, generated by the Tangle component of FunnelWeb, that contains the expansion of the macros in the input file. (Other names considered for this were: generated file, expanded file, result file, program file, and tangle file.)

A complete list of all the special FunnelWeb terminology appears in the glossary. Be sure to refer to it if any of the terms used are unclear.

Tutorial

Developer

Reference
1 Introduction
2 Interface
3 Scanner
4 Parser
5 Analyser
6 Tangle
7 Weave
8 Shell
9 Commands
10 Glossary
11 References

SEARCH

FunnelWeb Reference Manual

1.3 An Architectural Overview

An understanding of the internals of FunnelWeb assists with understanding its operation. During a single run, FunnelWeb reads and processes a single input file called the **input file** (also called the **FunnelWeb file**). The file is processed by passing it through a series of stages called **phases**. The result is that some **output files** are generated. A **journal file** is generated containing a copy of the messages that appear on the console during the FunnelWeb run. A **listing file** is created containing a summary of the run, including any error messages. A **documentation** file is generated containing typesetter commands that when fed into a typesetter program will result in printed documentation. Finally, one or more **product files** are generated containing the result of unscrambling the macro definitions of the input file.

```
     .fw Input File
     (FunnelWeb file)
            V
       +---------+        \
       | Scanner |        |
       +---------+        |
            V             |
        +--------+        |
        | Parser |        |
        +--------+        |
            V             |
       +----------+       >----+--------+
       | Analyser |       |    |        |
       +----------+       |    |        |
            V             |    |        |
    +-------+-------+     |    V        V
    V               V     |    |        |
+--------+      +-------+ |    |        |
| Tangle |      | Weave | |    |        |
+--------+      +-------+ /    |        |
    |               |          |        |
    V               V          V        V
 Product      Documentation Listing  Journal
  Files           File        File    File
```

FunnelWeb's processing phases

These files need not all be generated on any particular FunnelWeb run. Whether each output file appears, is controlled by command line options.

FunnelWeb processes each input file in a sequence of phases. If an error occurs during a phase, no subsequent phases are executed.

The phases are briefly described below.

> **The Scanner** reads the input file, expands and reads in include files, scans the input stream, processes pragmas and typesetter directives, and parses all the FunnelWeb special sequences. The result is a list of tokens that is handed to the parser.
>
> **The Parser** reads the scanner's token list and parses it, constructing a document list and a macro table. which are passed to later phases.
>
> **The Analyser** examines the macro table generated by the parser and performs a number of checks of the macro structures that the parser could not make on its single pass. For example, the analyser detects and flags unused macros and recursive macros. The analyser forms the final stage of FunnelWeb's front-end processing.
>
> **Tangle** expands certain macros in the macro table to generate one or more product files.
>
> **Weave** uses the document list to generate a documentation file.

A single run through these phases constitutes a single invocation of **FunnelWeb proper**. Most invocations of the **FunnelWeb program** will consist only of a single execution of FunnelWeb proper. However, FunnelWeb also provides a command shell that provides many useful commands, including a command to invoke FunnelWeb proper. Discussion of the command shell is deferred until later.

FunnelWeb Reference Manual

1.4 Diagnostics

During execution, FunnelWeb proceeds cautiously with each of its phases, only proceeding with the next phase if the previous phase has been successful. This means that, when debugging a FunnelWeb file, you may find that the number of errors *increases* after you fix some of them, as you will be exposing yourself to the next FunnelWeb phase.

FunnelWeb employs five levels of diagnostics at different levels of severity. Severity is defined in terms of the level of activity at which the diagnostic causes FunnelWeb to abort.

Warning: A warning does not cause FunnelWeb to terminate or curtail its operation in any way, but serves merely to warn the user of particular conditions that might be symptomatic of deeper problems.

Error: An error causes FunnelWeb to terminate processing of the current input file at the end of the current phase. For example, if an error occurs during scanning, FunnelWeb will continue scanning (and possibly generate further scanning diagnostics), but will not invoke the parser.

Severe Error: A severe error (or "severe" for short) is the same as an error except that FunnelWeb terminates the current phase immediately.

Fatal Error: A fatal error causes FunnelWeb not only to terminate the current phase and run immediately, but also to terminate total FunnelWeb processing immediately. A severe error will not cause a FunnelWeb script to terminate, but a fatal error will. A fatal error causes FunnelWeb to return control to the operating system.

Assertion Error: An assertion error occurs if FunnelWeb detects an internal inconsistency, in which case FunnelWeb terminates immediately and ungracefully. Such an error can occur only if there are bugs in FunnelWeb. With luck, such errors will be extremely rare.

FunnelWeb indicates the level of severity of each diagnostic that it issues by starting each diagnostic either with the full name of the severity level or with just the first letter of the severity level followed by a colon.

FunnelWeb conveys the presence or absence of diagnostics at the operating system level by returning EXIT_SUCCESS status if no diagnostics occurred during the run and EXIT_FAILURE status if one or more diagnostics (including warnings) occurred during the run. (From the symbols of the ANSI standard C library stdlib.h. See **[Kernighan88]**, p.252.)

FunnelWeb Reference Manual

1.5 Typesetter Independence

One of the design goals of FunnelWeb was to provide a *target-language independent* literate programming system. This goal has been achieved simply by treating the text written to the product file as homogeneous and typesetting it in tt font. A secondary goal was to provide a *typesetter independent* literate programming system. By this is meant that it be possible to create FunnelWeb input files that do not contain typesetter-specific commands. To a lesser extent this goal has also been achieved.

The difficulty with providing typesetter-independent typesetting is that each desired typesetting feature must be recreated in a typesetter-independent FunnelWeb typesetting construct that FunnelWeb can translate into whatever typesetting language is being targeted by Weave. Taken to the extreme, this would result in FunnelWeb providing the full syntactic and semantic power of TeX, but with a more generic, FunnelWeb-specific syntax. This was unfeasible in the time available, and undesirable as well.

The compromise struck in the FunnelWeb design is to provide a set of primitive typesetter-independent typesetting features that are implemented by FunnelWeb. These are the **typesetter directives**. If the user is prepared to restrict to these directives, then the user's FunnelWeb document will be both target-language and typesetter independent. However, if the user wishes to use the more sophisticated features of the target typesetting system, the user can specify the typesetter in a `"typesetter"` pragma and then place typesetter commands in the free text of the FunnelWeb document where they will be passed verbatim to the documentation file. The choice of the trade-off between typesetter independence and typesetting power is left to the user.

This said, experience with FunnelWeb, over more than a decade, indicates that the typesetting facilities provided by FunnelWeb are sufficient for most documentation.

FunnelWeb Reference Manual

2 Command Line Interface

FunnelWeb Reference Manual

2.1 Invoking FunnelWeb

When a user invokes FunnelWeb at the operating system command level, the user must provide a command line instructing FunnelWeb what to do. Typically an operating system command line consists of a *verb* indicating that a particular program should be run, followed by a list of options. For example:

```
rename file1 file2
```

In this case, the verb is rename and the command line options are file1 file2.

Operating systems differ greatly in the depth with which they process their command lines, ranging from systems that simply pass the entire command line string to the invoked program (e.g. MSDOS) through to systems that perform complete command line parsing (e.g. OpenVMS). Syntax conventions vary considerably.

So as to achieve maximum portability and consistency of invocation across different platforms, FunnelWeb reads its command line as a raw string and performs all its own parsing. This is portable because, at the very least, all operating systems allow invoked programs access to the raw command line.

The command verb used to invoke FunnelWeb should be "`fw`".

```
FunnelWeb_verb =  "fw"
```

If this verb is not available, some alternatives are "`funweb`", "`fun`", and "`funnelweb`". The verbs web or fweb should be avoided as they are the names of other literate programming tools.

FunnelWeb Reference Manual

2.2 Command Line Arguments

Following the verb is the body of the command line which FunnelWeb parses into zero or more **arguments** separated by runs of one or more blanks.

```
FunnelWeb_command_line =
   FunnelWeb_verb { {" "}+ argument }
```

Because some operating systems convert their command line to upper case before handing it to the invoked program, FunnelWeb has been constructed so as to be *insensitive* to the case of its command line arguments. However, when dealing internally with arguments, FunnelWeb *preserves* the case of its command line arguments so that it will be able to operate with operating systems (such as Unix) whose file names are case dependent.

A valid FunnelWeb argument consists of a **sign**, an identifying **letter**, and an optional **string** with no spaces separating them.

```
argument  = sign id_letter [non_blank_string]
sign      = "+" | "-" | "="
id_letter = "B" | "C" | "D" | "F" | "H" | "I" |
            "J" | "K" | "L" | "O" | "Q" | "S" |
            "T" | "W" | "X"
```

In addition there is a special form of argument that does not begin with a sign.

```
argument = non_blank_string_not_starting_with_+_=_or_-
```

This form is exactly equivalent to the same string with "+F" prepended to it.

The semantic effect of these arguments is defined in terms of **options** which are the internal parameters of FunnelWeb and which correspond closely with the set of legal command line arguments. FunnelWeb has a predefined set of options each identified by an identifying letter having two attributes: a *string* , and a *boolean* . The boolean determines whether an option is turned *on* or *off* . The string contains additional information depending on the option.

When FunnelWeb starts up, its options have predefined default values. FunnelWeb then parses its command line sequentially from left to right executing the effect of each argument on the argument's corresponding option. The sign and the string components of the argument are processed *independently* . A sign of + turns the option on. A sign of - turns the option off. A sign of = leaves the option's boolean attribute unchanged. The argument string replaces the string of the corresponding option, unless the argument string is empty, in which case the option string is not changed.

Because FunnelWeb processes its command line arguments from left to right, a later argument can cancel the effect of an earlier one. For example fw +t -t will result in the t option ending up *off* . This allows users to set up their own default arguments by defining a symbol in their operating system's command language. For example, a Unix user who wants FunnelWeb to delete all identical output files and create a documentation file on each run with a default .typ extension could simply place the following definition in their "`.login`" file.

```
alias fw fw +d +t.typ
```

These default options can then later be easily overridden on the command line.

FunnelWeb Reference Manual

2.3 Options

FunnelWeb's options are internal parameters which can be modified by corresponding arguments on FunnelWeb's command line. A description of each argument and option follows.

> **B1...B6: Tracedumps:** These six options have been provided to assist in the debugging and testing of FunnelWeb. They determine which of six possible trace dumps are to be written to the listing file. Only the boolean attributes of these options are ever used. The six dumps are identified by the digits 1..6 as follows:

1. Dump a hexdump of each mapped input and include file.
2. Dump the global line list created by the scanner.
3. Dump the token list created by the scanner.
4. Dump the macro table created by the parser.
5. Dump the document list created by the parser.
6. Dump a table summarizing CPU and real time usage.

Because these options are so closely related, a hack has been pulled to enable them to all to be controlled by the B argument. The string argument to the B argument determines which of the six options are to be affected by the sign. Examples: +B134 turns on options B1, B3, and B4. -B1 turns off option B1. Default: -B123456.

> **B7: Determinism:** If the B7 option is turned on, FunnelWeb suppresses the output of anything non-deterministic, or machine dependent. This assists in regression testing. Only the boolean attribute is used in this option. This option is controlled by the B7 argument which falls under the same argument syntax as the other B options. Examples: +B7, -B7. Default: -B7.

> **C: Listing File Context:** The C option is always turned on and cannot be turned off. Its only attribute is a number which determines the number of lines of context that the lister will place around lines flagged with diagnostics in the listing file (if a listing file is written). A value of 100 indicates infinite context which means that the entire listing

file will be written out if a single diagnostic occurs. The value of this number can be specified by specifying it as a string of decimal digits to the +C argument. Examples: +C100, +C10.
Default: +C2.

D: Delete Identical Output Files: Only the boolean attribute of this option is used. When turned on, the option causes the suppression (deletion) of product files and documentation files (but not listing or journal files) that are identical to the currently existing files of the same name. For example, if FunnelWeb is instructed to generate stack.h as an product file, and the text to be written to stack.h is identical to the currently existing stack.h, then FunnelWeb will simply not write any product file, leaving the currently existing stack.h as it is (and in particular leaving the file's date attribute the same). This prevents unnecessary make propagations. For example, in a C program, if stack.fw is a FunnelWeb input file that generates stack.h and stack.c, a modification to stack.fw that affects stack.c but does not affect stack.h will not provoke the recompilation of modules that #include stack.h, so long as the intervening FunnelWeb run has +D set. Examples: -D, +D.
Default: -D.

F: FunnelWeb Input File: If this option is turned on, FunnelWeb processes the input file whose name is specified by the option string. Examples: +Fsloth.fw, +Fwalrus, -F. Default: -F.

H: Display Help Message: If this option is turned on, FunnelWeb displays the message specified by the argument string. Each message has a name. The main help message is called "`menu`" and contains a list of the other help messages. Examples: +Hregistration, +Hoptions.
Default: -Hmenu.

I: Include default file specification: This option is always turned on and cannot be turned off. Its string attribute is used as the default file specification for include files. Usually this option is used to specify a directory from which include files should be obtained. Examples: =I/usr/dave/includes/. Default: +I.

J: Journal File: If this option is turned on, FunnelWeb generates a journal file. A journal file contains a log of all the console input and output to FunnelWeb during a single invocation of the FunnelWeb program (Note: The Q option does not affect this.). The journal file is particularly useful for examining what happened during a FunnelWeb shell run. The string attribute is the name of the journal file. Examples: +Jjournfile, -J. Default: -J.

K: Keyboard: If this option is turned on, FunnelWeb enters an interactive mode in which the user can enter FunnelWeb shell commands interactively. The string attribute is unused. Examples: +K, -K. Default: -K.

L: Listing File: If this option is turned on, FunnelWeb generates a listing file containing a summary of a run on FunnelWeb proper. The string argument is the name of the listing file to be created. Examples: +L, -L, +Llisting.lis. Default: -L.

O: Product Files: If this option is turned on, FunnelWeb generates a product file for each macro in the input file that is bound to an output file. The string attribute contributes to the name of the product files. This option is controlled by the O argument because product files used to be called "**O**utput files"). Examples: -O, +O/usr/dave/product/. Default: +O.

Q: Quiet: If this option is turned on, FunnelWeb suppresses all output to the screen (standard output) unless one or more errors occur, in which case a single line summarizing the errors is sent to standard output at the end of the run. If this option is turned off, FunnelWeb writes to the console in its normal garrulous way. The string attribute is unused in this option. Examples: -Q, +Q. Default: -Q.

S: Screen: If this option is turned on, FunnelWeb writes all diagnostics to the screen (standard output) as well as to the listing file. By default, they are sent only to the listing file. This option has a single numerical attribute that can be specified as a decimal string in the string component of the S argument. The number is the number of lines of context that should surround each diagnostic sent to the screen. Examples: -S, +S6, +S0. Default: -S.

T: TeX Documentation file: If this option is turned on, FunnelWeb generates a documentation file in TeX format. The string argument contributes to the name of the documentation file to be created. By default this option is turned off, as experience has shown that most FunnelWeb runs are made during program development; documentation runs occur far more rarely. This option is controlled by the T command line argument because documentation files used to be called typesetter files. Examples: -T, +Tsloth.tex. Default: -T.

U: HTML Documentation file: If this option is turned on, FunnelWeb generates a documentation file in HTML format. The string argument contributes to the name of the documentation file to be created. By default this option is turned off, as experience has shown that most FunnelWeb runs are made during program development; documentation runs occur far more rarely. U was chosen for this feature because it follows T, and H and W were already allocated to other functions. Examples: -U, +Usloth.html. Default: -U.

W: Width of Product Files: If this option is turned on, a limit is placed on the length of lines in product files generated during the run. Lines that breach the limit are flagged with error messages. This option has a single numerical attribute that can be specified as a decimal string in the string component of the W argument. The number is the specified maximum width. This option is one of two limits that are placed on the width of product files. The other limit is an attribute of the input file that defaults to 80 characters, but can be raised or lowered using an output line length pragma. The width that is enforced is the lower of this value and the value of the W option (if turned on). Examples: -W, +W100. Default: -W80.

X: Execute: If this option is turned on, FunnelWeb executes the FunnelWeb shell script file specified by the string attribute. Examples: +Xmaster, -X. Default: -X.

FunnelWeb Reference Manual

2.4 File Name Inheritance

During a single run of FunnelWeb, FunnelWeb can produce many different output files. As it would be very tedious to have to specify the name of each of these files explicitly each time FunnelWeb is run, FunnelWeb provides a system of defaults that allows the user to specify the minimum required to successfully complete the run. To do this, FunnelWeb allows file specifications to inherit fields from one another.

FunnelWeb structures filenames into three fields which are inherited independently. The fields are: **directory**, **name**, and **extension**. On systems having other fields (e.g. *network node* , *device name*), the extra fields are considered to be part of the directory field. Version numbers are ignored. A field can inherit a value if its current value is the empty string.

The following table gives the full inheritance scheme used in FunnelWeb.

Script	Input	Include	Journal	List	TeX	HTML	Prod
		@i					@o
+x	+f	+i	+j	+l	+t	+u	+o
".fws"	".fw"	".fwi"	".jrn"	".lis"	".tex"	".html"	
		+f	+f	+f	+f	+f	
DefDir	Defdir	Defdir	Defdir	Defdir	Defdir	Defdir	Defdir

The table is arranged with items of highest priority at the top. The "+" cells refer to the file specification supplied in the given command line argument. "+F" is the name of the input file. "Defdir" refers to the default directory specification provided by the operating system. Empty cells do not contribute.

The following example shows how the table is used. Suppose that the user invoked FunnelWeb as follows:

```
fw /usr/ross/work/sloth.fw +twalrus
```

To work out what the documentation file should be called, FunnelWeb starts with the empty string and then works down the Document column of the table. The top entry is empty so we ignore it and proceed to the second entry which consists of "+T". The user specified the string "walrus" as the value of this option, and as our current (empty) string does not have a name field, we insert the string "`walrus`" into the name field, resulting in the string "`walrus`". Moving down to the next row, we encounter the constant string "`.tex`". This string consists of an empty directory and name field, but a "`.tex`" file extension. As our current string "`walrus`", does not already have a file extension (i.e. the file extension field of our current string is empty), we add in "`.tex`", resulting in the string "`walrus.tex`". Next we encounter the "`+F`" field which is the input filename "`/usr/ross/work/sloth.fw`" consisting of a directory field "`/usr/ross/work/`", a name field "`sloth`", and a file extension field "`.fw`". Our "`walrus.tex`" string already has name and file extension fields, but its directory field is empty, and so we add in the directory field from the input file specification, resulting in the string "`/usr/ross/work/walrus.tex`". Finally, we hit the default directory specification, which is (say) "`/usr/ross/play/`". However, as the directory field of our walrus string is already full, it has no effect.

In general, there is no need to remember the exact details of FunnelWeb's filename inheritance. The important thing is to know that it exists, and to use it.

FunnelWeb Reference Manual

2.5 FunnelWeb Startup

FunnelWeb's command line options can be divided into two groups. **Action options** instruct FunnelWeb to performs some sort of independent action such as processing a file. **Ordinary options** merely modify the way in which FunnelWeb executes the actions.

The four action options are: +F, +K, +X, and +H.

When FunnelWeb is invoked, if no action options are specified, and there is a file called fwinit.fws in the current directory, then FunnelWeb executes the FunnelWeb script file (.fws) and terminates. If there isn't such a file, FunnelWeb issues a help message and terminates.

If, when FunnelWeb is invoked, one or more action options are specified, FunnelWeb performs the actions in a predefined order as follows:

> **Initialization script:** FunnelWeb starts by looking in the current directory for a file called "`fwinit.fws`". If it doesn't find one, it doesn't raise any error. If it does find one, it executes it as a FunnelWeb shellscript. Initialization scripts are useful for setting up FunnelWeb options (e.g. using the "`set`" command without having to type them each time).
>
> **Execute argument script:** If a shellscript has been specified using the "`+X`" option, FunnelWeb executes it.
>
> **Process input file:** If the user has specified an input file using the "`+F`" option, then this is processed next (by FunnelWeb proper).
>
> **Display help message:** If the user requested, using the "`+H`" option, that a help message be displayed, the message is displayed at this time.
>
> **Interactive mode:** If the user specified the "`+K`" option, FunnelWeb enters interactive (keyboard) mode.

FunnelWeb processes these actions in the above order regardless of the order in which they appear on the command line.

It may be hard to see how some of these actions might be combined. Nevertheless, FunnelWeb allows this. For example, a user might wish to process a batch of files as specified in a script ("`+Xscript.fws`"), be reminded of the interactive commands available ("`+Hcommand`"), and then enter interactive mode so as to be able to reprocess files for which FunnelWeb reported errors (after correcting the errors in a different workstation window).

FunnelWeb Reference Manual

3 Scanner

The scanner reads in the input file and produces a list of tokens which it hands onto the parser. In addition, some input constructs may cause the scanner to modify some of FunnelWeb's options.

FunnelWeb Reference Manual

3.1 Basic Input File Processing

In order to read in an input file or include file, the scanner calls a submodule called the **mapper** that reads a file in and creates a contiguous copy of it in memory. The scanner then performs three checks on the file, the first (file termination) of which is performed before scanning commences, and the other two of which take place during scanning before each line is scanned.

File Termination: The first check the scanner makes is whether the file is terminated properly. A file is considered to be properly terminated if it either contains no lines, or if the last line in the file is terminated by an end-of-line marker. If the scanner detects that an input file is not properly terminated, it adds an end-of-line marker itself (to the copy in memory only).

Unprintable Characters: The second check the scanner makes is for unprintable characters (ASCII 0--31 and 127--255 (except for EOL(10))) which it flags as errors and replaces by question marks.

Line Lengths: The third check the scanner makes is input line length. When FunnelWeb starts up, a default maximum input line length of 80 is set. This can be changed dynamically during scanning using a @p maximum_input_line_length pragma. If the number of characters on a line (not including the end of line marker) exceeds this limit, FunnelWeb generates an error.

FunnelWeb Reference Manual

3.2 Special Sequences

The scanner scans the input file from top to bottom, left to right, treating the input as ordinary text (to be handed directly to the parser as a text token) unless it encounters the **special character** which introduces a **special sequence**. Thus, the scanner partitions the input file into ordinary text and special sequences. (Note: This sort of character is often referred to as the "escape character" or the "control character" in other systems. However, as there is great potential to confuse these names with the "escape" character (ASCII 27) and ASCII "control" characters, the term "special" has been chosen instead. This results in the terms *special character* and *special sequence* .

```
input_file = { ordinary_text | special_sequence }
```

Upon startup, the special character is @, but it can be changed using the <special>=<new_special> special sequence. Rather than using <special> whenever the special character appears, this document uses the default special character "@" to represent the current special character. More importantly, FunnelWeb's error messages all use the default special character in examples, even if the special character has been changed.

An occurrence of the special character in the input file introduces a special sequence. The kind of special sequence is determined by the character following the special character. Only printable characters can follow the special character.

The following list gives all the possible characters that can follow the special character, and the legality of each sequence. The first column gives the ASCII number of each ASCII character. The second column gives the special sequence for that character. The next column contains one of three characters: "-" means that the sequence is illegal. "S" indicates that the sequence is a **simple sequence** (with no attributes or side effects) that appears exactly as shown and is converted directly into a token and fed to the parser. Finally, "C" indicates that the special sequence is complex, possibly having a following syntax or producing funny side effects.

```
ASC  SEQ  COMMENT
-----------------
000       \
016        | Unprintable chars (illegal specials).
031       /
032  @    - Illegal (space).
033  @!   C Comment.
034  @"   S Parameter delimeter.
```

```
035  @#   C Short name sequence.
036  @$   S Start of macro definition.
037  @%   - Illegal.
038  @&   - Illegal.
039  @'   - Illegal.
040  @(   S Open parameter list.
041  @)   S Close parameter list.
042  @*   - Illegal.
043  @+   C Insert newline.
044  @,   S Parameter separator.
045  @-   C Suppress end of line marker.
046  @.   - Illegal.
047  @/   S Open or close emphasised text.
048  @0   - Illegal.
049  @1   S Formal parameter 1.
050  @2   S Formal parameter 2.
051  @3   S Formal parameter 3.
052  @4   S Formal parameter 4.
053  @5   S Formal parameter 5.
054  @6   S Formal parameter 6.
055  @7   S Formal parameter 7.
056  @8   S Formal parameter 8.
057  @9   S Formal parameter 9.
058  @:   - Illegal.
059  @;   - Illegal.
060  @<   S Open macro name.
061  @=   C Set special character.
062  @>   S Close macro name.
063  @?   - Illegal. Reserved for future use.
064  @@   C Insert special character into text.
065  @A   S New section (level 1).
066  @B   S New section (level 2).
067  @C   S New section (level 3).
068  @D   S New section (level 4).
069  @E   S New section (level 5).
070  @F   - Illegal.
071  @G   - Illegal.
072  @H   - Illegal.
073  @I   C Include file.
074  @J   - Illegal.
075  @K   - Illegal.
076  @L   S Macro is library macro.
077  @M   S Macro may be called many times.
078  @N   - Illegal.
079  @O   S Macro is attached to product file.
080  @P   C Pragma.
```

```
081  @Q   - Illegal.
082  @R   - Illegal.
083  @S   - Illegal.
084  @T   C Typesetter directive.
085  @U   - Illegal.
086  @V   - Illegal.
087  @W   - Illegal.
088  @X   - Illegal.
089  @Y   - Illegal.
090  @Z   S Macro may be called zero times.
091  @[   - Illegal. Reserved for future use.
092  @\   - Illegal.
093  @]   - Illegal. Reserved for future use.
094  @^   C Insert control character into text
095  @_   - Illegal.
096  @`   - Illegal.
097  @a   \
109  @m    | Identical to @A..@Z.
122  @z   /
123  @{   S Open macro body/Open literal directive.
124  @|   - Illegal.
125  @}   S Close macro body/Close literal directive.
126  @~   - Illegal.
127 to 255 - Illegal specials (as non-standard ASCII).
```

The most important thing to remember about the scanner is that *nothing happens unless the special character is seen.* There are no funny sequences that will cause strange things to happen. The best way to view a FunnelWeb document at the scanner level is as a body of text punctuated by special sequences that serve to structure the text at a higher level.

The remaining description of the scanner consists of a detailed description of the effect of each complex special sequence.

FunnelWeb Reference Manual

3.3 Setting the Special Character

The special character can be set using the sequence <special>=<newspecialchar>. For example, @=# would change the special character to a hash (#) character. The special character may be set to any printable ASCII character except the blank character (i.e. any character in the ASCII range [33,126]). In normal use, it should not be necessary to change the special character of FunnelWeb, and it is probably best to avoid changing the special character so as not to confuse FunnelWeb readers conditioned to the @ character. However, the feature is very useful where the text being prepared contains many @ characters (e.g. a list of internet electronic mail addresses).

FunnelWeb Reference Manual

3.4 Inserting the Special Character into the Text

The special sequence <special>@ inserts the special character into the text as if it were not special at all. The @ of this sequence has nothing to do with the current special character. If the current special character is P then the sequence P@ will insert a P into the text. Example: @@#@=#@#@#=@@@ translates to @#@#@.

FunnelWeb Reference Manual

3.5 Inserting Arbitrary Characters into the Text

While FunnelWeb does not tolerate unprintable characters in the input file (except for the end of line character), it does allow the user to specify that unprintable characters appear in the product file. The @^ sequence inserts a single character of the user's choosing into the text. The character can be specified by giving its ASCII number in one of four bases: binary, octal, decimal, and hexadecimal. Here is the syntax:

```
control_sequence = "@^" char_spec
char_spec        = binary  | octal |
                   decimal | hexadecimal
binary           = ("b" | "B")
                   "(" {binary_digit}8  ")"
octal            = ("o" | "O" | "q" | "Q")
                   "(" {octal_digit}3   ")"
decimal          = ("d" | "D")
                   "(" {decimal_digit}3 ")"
hexadecimal      = ("h" | "H" | "x" | "X")
                   "(" {hex_digit}2     ")"
binary_digit     = "0" | "1"
octal_digit      = binary_digit | "2" | "3" |
                   "4" | "5" | "6" | "7"
decimal_digit    = octal_digit | "8" | "9"
hex_digit        = decimal_digit |
                   "A" | "B" | "C" | "D" | "E" | "F" |
                   "a" | "b" | "c" | "d" | "e" | "f"
```

Example:

```
@! Unix Make requires that productions
@! commence with tab characters.
@^D(009)prog.o <- prog.c
```

Note that the decimal "`9`" is expressed with leading zeros as "`009`". FunnelWeb requires a fixed number of digits for each base. Eight digits for base two, three digits for base ten, three digits for base eight and two digits for base sixteen.

FunnelWeb treats the character resulting from a @^ sequence as ordinary text in every sense. If your input file contains many instances of a particular control character, you can package it up in a macro like any other text. In particular, quick names can be used to great effect:

```
@! Unix "Make" requires that productions
@! commence with tab characters.
@! So we define a macro with a quick name
@! as a tab character.
$@#T@{@^D(009)@}
@! And use it in our productions.
@#Tprog.o <- prog.c
@#Ta.out <- prog.o
```

Warning: If you insert a Unix newline character (decimal 10) into the text, FunnelWeb will treat this as an end of line sequence regardless of what the character sequence for end of line is on the machine upon which it is running. Unix EOL is FunnelWeb's internal representation for end of line. Thus, in the current version of FunnelWeb, inserting character 10 into the text is impossible unless this also happens to be the character used by the operating system to mark the end of line.

FunnelWeb Reference Manual

3.6 Comments

When FunnelWeb encounters the @! sequence during its left-to-right scan of the line, it throws away the rest of the line (including the EOL) without analysing it further. Comments can appear in any line except "`@i`", "`@t`", and "`@p`" lines.

FunnelWeb comments can be used to insert comments into your input file that will neither appear in the product files nor in the documentation file, but will be solely for the benefit of those reading and editing the input file directly. Example:

```
@! I have used a quick macro for this definition
@! as it will be used often.
@$@#C@{--@}
```

Because comments are defined to include the end-of-line marker, care must be taken when they are being added or removed within the text of macro bodies. For example the text fragment

```
for (i=0;i<MAXVAL;i++) @! Print out a[0..MAXVAL-1].
   printf("%u\n",a[i]);
```

will expand to

```
for (i=0;i<MAXVAL;i++) printf("%u\n",a[i]);
```

This problem really has no solution; if FunnelWeb comments were defined to omit the end of line marker, the expanded text would contain trailing blanks! As it is, FunnelWeb comments are designed to support single line comments which can be inserted and removed as a line without causing trouble. For example:

```
@! Print out a[0..MAXVAL-1].
for (i=0;i<MAXVAL;i++)
   printf("%u\n",a[i]);
```

If you want a comment construct that does not enclose the end of line marker, combine the insert end of line construct @+ with the comment construct @! as in

```
for (i=0;i<MAXVAL;i++)  @+@! Print out a[0..MAXVAL-1].
   printf("%u\n",a[i]);
```

FunnelWeb comments should really only be used to comment the FunnelWeb constructs being used in the input file. Comments on the target code are best placed in comments in the target language or in the documenting text surrounding the macro definitions. In the example above, a C comment would have been more appropriate.

FunnelWeb Reference Manual

3.7 Quick Names

FunnelWeb provides a **quick name** syntax as an alternative, for macros whose name consists of a single character, to the angle bracket syntax usually used (e.g. @<Sloth@>). A quick name sequence consists of @#x where x, the name of the macro, can be any printable character except space.

```
quick_name = "@#" non_space_printable
```

The result is identical to the equivalent ordinary name syntax, but is shorter. For example, @#X is equivalent to @<X@>. This shorter way of writing one-character macro names is more convenient where a macro must be used very often. For example, the macro calls in the following fragment of an Ada program are a little clumsy.

```
@! Define @<D@> as "" to turn on debug code
@! and "--" to turn it off.
@$@<D@>@{--@@)
@<D@>assert(b>3);
@<D@>if x>7 then write("error") end if
```

The calls can be shortened using the alternative syntax.

```
@! Define @#| as "" to turn on debug code
@! and "--" to turn it off.
@$@#|@{--@@)
@#|assert(b>3);
@#|if x>7 then write("error") end if
```

FunnelWeb Reference Manual

3.8 Inserting End of Line Markers

An end of line marker/character can be inserted into the text using the @+ sequence. This is exactly equivalent to a real end of line in the text at the point where it occurs. While this feature may sound rather useless, it is very useful for laying out the input file. For example, the following input data for a database program

```
Animal = Kangaroo
Size   = Medium
Speed  = Fast

Animal = Sloth
Size   = Medium
Speed  = Slow

Animal = Walrus
Size   = Big
Speed  = Medium
```

can be converted into

```
Animal = Kangaroo @+Size = Medium @+Speed = Fast   @+
Animal = Sloth    @+Size = Medium @+Speed = Slow   @+
Animal = Walrus   @+Size = Big    @+Speed = Medium @+
```

which is easier to read, and more easily allows comparisons between records.

FunnelWeb Reference Manual

3.9 Suppressing End of Line Markers

End of line markers can be suppressed by the @- sequence. A single occurrence of a @- sequence serves to suppress only the end of line marker following it and must appear *exactly* before the end of line marker to be suppressed. No trailing spaces, @! comments, or any other characters are permitted between a @- sequence and the end of line that it is supposed to suppress. The @- sequence is useful for constructing long output lines without them having to appear in the input. It can also be used in the same way as the @+ was used in the previous section to assist in exposing the structure of output text without affecting the output text itself. Finally, it is invaluable for suppressing the EOL after the opening macro text @{ construct. For example:

```
@$@<Walrus@>@{@-
I am the walrus!@}
```

is equivalent to

```
@$@<Walrus@>@{I am the walrus!@}
```

The comment construct (@!) can also be used to suppress end of lines. However, the @- construct should be preferred for this purpose as it makes explicit the programmer's intent to suppress the end of line.

3.10 Include Files

FunnelWeb provides an include file facility with a maximum depth of 10. When FunnelWeb sees a line of the form @i <filename>, it replaces the entire line (including the EOL) with the contents of the specified include file. FunnelWeb's include file facility is intended to operate at the line level. If the last line of the include file is not terminated by an EOL, FunnelWeb issues a warning and inserts one (in the copy in memory).

The @i construct is illegal if it appears anywhere except at the start of a line. The construct must be followed by a single blank. The file name is defined to be everything between the blank and the end of the line (no comments (@!) please!). Example: If the input file is

```
"Uh Oh, It's the Fuzz. We're busted!" said Baby Bear.
@i mr_plod.txt
"Quick! Flush the stash down the dunny and split."
   said Father Bear.
```

and there is a file called mr_plod.txt containing

```
"'Ello, 'Ello, 'Ello! What's all this 'ere then?"
   Mr Plod exclaimed.
```

then the scanner translates the input file into

```
"Uh Oh, It's the Fuzz. We're busted!" said Baby Bear.
"'Ello, 'Ello, 'Ello! What's all this 'ere then?"
   Mr Plod exclaimed.
"Quick! Flush the stash down the dunny and split."
   said Father Bear.
```

As a point of terminology, FunnelWeb calls the original input file the **input file** and calls include files and their included files **include files**.

The include file construct operates at a very low level. An include line can appear anywhere in the input file regardless of the context of the surrounding lines.

FunnelWeb sets the special character to the default (@) at the start of each include file and restores it to its previous value at the end of the include file. This allows macro libraries to be constructed and included that are independent of the prevailing special character at the point of inclusion. The same goes for the input line length limit which is reset to the default value at the start of each include file and restored to its previous value afterwards.

FunnelWeb Reference Manual

3.11 Pragmas

Most tools have to support some essential, but rather inelegant features. In FunnelWeb these messy bits have all been stuffed into the scanner's **pragma** (for *pragma* tic) construct.

A pragma consists of a single line of input (including the EOL) commencing with @p. This must be followed by a single space, and then the pragma verb. This must be followed by a sequence of zero or more arguments separated by one or more spaces. Four pragmas are available

```
pragma = pragma_ident | pragma_mill |
         pragma_moll | pragma_typesetter
```

The following syntax definitions assist in defining the pragmas.

```
s         = {" "}+
ps        = ("@p" | "@P") " "
number    = { decimal_digit }+
numorinf = number | "infinity"
```

The arguments to pragmas are case-sensitive and must be specified in lower case.

Pragmas are processed and consumed entirely by the scanner. The parser never sees them and so they can play no part in the parser level syntax. As a result, pragma lines can appear anywhere in the entire input file regardless of the surrounding context (e.g. even in the middle of a macro definition). The sole effect of a pragma is to modify some internal parameter of FunnelWeb.

The following sections describe the four FunnelWeb pragmas.

FunnelWeb Reference Manual

3.12 Pragma: Indentation

When FunnelWeb expands a macro, it can do so in two ways. First it can treat the text it is processing as a one-dimensional stream of text, and merely insert the body of the macro in place of the macro call. Second, it can treat the text of the macro as a two dimensional object and indent each line of the macro body by the amount that the macro call itself was indented. Consider the following macros.

```
@$@<Loop Structure@>@{@-
i=1;
while (i<=N)
   @<Loop body@>
endwhile
@}

@$@<Loop body@>@{@-
a[i]:=0;
i:=i+1;@}
```

Under the regime of **no indentation** the loop structure macro expands to:

```
i=1;
while (i<=N)
   a[i]:=0;
i:=i+1;
endwhile
```

Under the regime of **blank indentation** the loop structure macro expands to:

```
i=1;
while (i<=N)
   a[i]:=0;
   i:=i+1;
endwhile
```

The indentation pragma determines which of these two regimes will be used to expand the macros when constructing the product files. The syntax of the pragma is:

```
pragma_ident = ps "indentation" s "=" s
               ("blank" | "none")
```

Its two forms look like this:

```
@p indentation = blank
@p indentation = none
```

In the current version of FunnelWeb, the indentation regime is an attribute that is attached to an entire run of Tangle; it is not possible to bind it to particular product files or to particular macros. As a result, it doesn't matter where indentation pragmas occur in the input file or how many there are so long as they are all the same. By default FunnelWeb uses blank indentation.

FunnelWeb Reference Manual

3.13 Pragma: Maximum Input Line Length

FunnelWeb generates an error for each input line that exceeds a certain maximum number of characters. At the start of the processing of each input file and each include file, this maximum is set to a default value of 80. However, the maximum can be changed using a maximum input line length pragma.

```
pragma_mill = ps "maximum_input_line_length" s
              "=" s numorinf
```

The maximum input line length can be varied *dynamically* throughout the input file. Each maximum input line length pragma's scope covers the line following the pragma through to and including the next maximum input line length pragma, but not covering any intervening include files. At the start of an include file, FunnelWeb resets the maximum input line length to the default value. It restores it to its previous value at the end of the include file.

This pragma is useful for detecting text that has strayed off the right side of the screen when editing. If you use FunnelWeb, and set the maximum input line length to be the width of your editing window, you will never be caught by, for example, off-screen opening comment symbols. You can also be sure that your source text can be printed raw, if necessary, without lines wrapping around.

FunnelWeb Reference Manual

3.14 Pragma: Maximum Output File Line Length

As well as keeping an eye on input line lengths, FunnelWeb also keeps an eye on the line lengths of product files and flags all lines longer than a certain limit with error messages. Unlike the maximum input line length, which can vary dynamically throughout the input file, the maximum product file line length remains fixed throughout the generation of all the product files. The maximum product file line length pragma allows this value to be set. If there is more than one such pragma in an input file, the pragmas must all specify the same value.

```
pragma_moll = ps "maximum_output_line_length"
              s "=" s numorinf
```

The default value is 80 characters.

This pragma is only one of two constraints on the length of the lines of the product files. The +W command line option also contributes. The actual value that FunnelWeb uses is the minimum of the limits specified in the command line and pragmas.

FunnelWeb does not monitor the length of the lines of its other output files (journal file, listing file, documentation file).

FunnelWeb Reference Manual

3.15 Pragma: Typesetter

The typesetter pragma allows the user to specify whether the input file is supposed to be typesetter-independent, or whether it contains commands in a particular typesetter language. The pragma has the following syntax.

```
pragma_typesetter = ps "typesetter" s "="
                    s ("none" | "tex" | "html")
```

The three forms of the pragma look like this.

```
@p typesetter = none
@p typesetter = tex
@p typesetter = html
```

A source file can contain more than one typesetter pragma, but they must all specify the same value. The default is none. The typesetter setting affects two things:

Handling of free text: If the typesetter is not none, Weave writes the free text *directly* to the documentation file without changing it whatsoever. This means that if (say) \centerline appears in the input file, it will copied directly to the documentation file. If the typesetter is none, Weave intercepts any characters or sequences that might have a special meaning to the target typesetter and replaces them with typesetter commands to typeset the sequences so that they will appear as they do in the input. For example, if the typesetter is none and the target typesetter is TeX, then if $ (the TeX "mathematics mode" character) appears in the input file, it will be be written to the documentation file as \$.

Restrictions on the target typesetter: If you make your FunnelWeb input file depedent on one particular typesetter, it's important that no attempt be made to generate documentation for a different target format. FunnelWeb enforces this at the Weave stage.

The aim of all this is to ensure that any typesetter dependency is correctly proclaimed. Because none is the default typesetter, a user who creates a source file without a typesetter = x pragma will soon find that the control sequences they are inserting into the source document are appearing verbatim in the printed documentation! In order to activate these sequences, they will be forced to add a typesetter pragma, thus making the dependency explicit.

It may seem strange to place the typesetter setting facility within a pragma (@p) when there is a separate typesetting construct (@t). This has been done to sustain the rule of thumb that says that pragmas do not participate in the parser-level syntax, but typesetter directives do.

FunnelWeb Reference Manual

3.16 Freestanding Typesetter Directives

FunnelWeb provides two kinds of typesetter directive to assist the user to produce documentation. These are **inline** and **freestanding**. Unlike pragmas, each of these categories of directive participates in the parser-level syntax and can appear only in certain contexts (see the parser section). Inline directives are designed to be used within paragraphs to alter the look of the enclosed text. Freestanding typesetter directives are designed to appear on lines of their own and have a bigger typographical impact.

The syntax of freestanding typesetter directives is almost identical to that of pragmas. All the same syntax rules apply (except that the actual keywords are different). The following subsections describe the four typesetter directives available.

```
ftd = ftd_newpage | ftd_toc | ftd_vskip | ftd_title
ts  = "@t "
```

FunnelWeb Reference Manual

3.17 New Page Directives

The new page pragma is a typesetting pragma with the following syntax.

```
ftd_newpage = ts "new_page"
```

It only form looks like this.

```
@t new_page
```

Its sole effect is to cause a "skip to a new page" command to be inserted into the documentation file. The new page command is such that if the typesetter is already at the top of a page, it will skip to the top of the next page.

FunnelWeb Reference Manual

3.18 Table of Contents

The table of contents pragma is a typesetting pragma with the following syntax.

```
ftd_toc = ts "table_of_contents"
```

It only form looks like this.

```
@t table_of_contents
```

Its sole effect is to instruct Weave to insert a table of contents at this point in the printed documentation. This pragma does not skip to a top of a new page first.

FunnelWeb Reference Manual

3.19 Vertical Skip

The vertical skip pragma is a typesetting pragma that instructs Weave to insert a specified amount of vertical space into the documentation. The pragma has the following syntax.

```
ftd_vskip = ts "vskip" s number s "mm"
```

For example:

```
@t vskip 26 mm
```

FunnelWeb Reference Manual

3.20 Title

The title pragma is a typesetting pragma with the following syntax.

```
ftd_title = ts "title" s font s alignment text
font      = "normalfont" | "titlefont" |
            "smalltitlefont"
alignment = "left" | "centre" | "right"
text      = """" { printable_char } """"
```

It's effect is to instruct Weave to insert a single line into the printed documentation containing the specified text set in the specified font and aligned in the specified manner. The double quotes delimiting the text are for show only; if you want to put a double quote in the string, you don't need to double them.

Here is an example of the pragma.

```
@t title smalltitlefont centre "How to Flip a Bit"
```

FunnelWeb Reference Manual

3.21 Scanner/Parser Interface

If the scanner terminates without any errors, control is passed to the parser. The parser parses the token list generated by the scanner. The token list consists of text scraps, freestanding typesetter directives, and special sequence tokens.

The user should bear in mind that *the scanner finishes running before the parser starts running.* This means that the scanner cannot be influenced in any way by higher order structures such as the parser might parse. For example, it is impossible to write a FunnelWeb macro to include a file, or to write a macro that inserts a vskip pragma into the input text.

FunnelWeb Reference Manual

4 Parser

FunnelWeb Reference Manual

4.1 Introduction

By the time the parser starts, the scanner has completely terminated. At this point, it is not possible for any more files to be included, and special characters are no longer present to confuse things. All that remains is a list of **text tokens**, **special tokens**, and **typesetter directive tokens**. Text tokens consist entirely of sequences of printable characters and end of line markers. Special tokens represent the special sequences that the scanner found in the input file. Typesetter directive tokens represent the freestanding typesetter directives that the scanner encountered. The parser consumes the token list and builds a macro table that is later used to generate product files. It also constructs a document list that is used to generate the documentation file.

The syntax rules appearing in the following sections refer to the token list.

FunnelWeb Reference Manual

4.2 High Level Structure

At the highest level, the FunnelWeb parser parses the input file (token list) into a sequence of text scraps, macro definitions, and typesetter directives.

```
input_file = { text | macro | directive }
```

All three of these kinds of components contribute to the documentation file, but only macro definitions contribute to the product files. If all the free text and directives were removed from a FunnelWeb input file, the product files would not be affected.

FunnelWeb Reference Manual

4.3 Free Text

Free text is any text that is not part of a macro definition or a directive. A scrap of free text consists of a sequence of items drawn from the following list: non-special printable characters, insert-eol special sequences, insert special character special sequences, insert arbitrary character special sequence.

```
free_text      = ordinary_text
ordinary_text  = { ordinary_char | eol | text_special }+
text_special   = "@+" | "@@" | "@^" char_spec
ordinary_char  = " ".."~" - special
```

An example of some rather messy free text is as follows:

```
This@@ is a very@+ messy
@^D(009)chunk of text indeed.
But FunnelWeb still views it as
a single chunk of text.
```

FunnelWeb never sees two text chunks next to each other in the input; they are always merged into a single text token.

The free text in an input file does not affect the product files. However, by default, it appears in the printed documentation exactly as it is given in the input file, except that it is filled and justified into paragraphs.

Any printable character or particular sequence of characters may appear in the free text of a document. FunnelWeb ensures that they will appear exactly as given in the input file, even if they happen to be escape characters or commands in the target typesetter. However, FunnelWeb also provides a special mode that allows this censoring to be overridden.

FunnelWeb Reference Manual

4.4 Typesetter Directives

FunnelWeb provides a variety of typesetter directives to assist the user to typeset the document in a typesetter-independent way. These are divided into **freestanding typesetter directives** (ftd) and **inline typesetter directives** (itd). The internal syntax of the freestanding typesetter directives has already been discussed in the scanner section. The following syntax rule defines the context in which these constructs can appear.

```
directive = ftd | itd
itd       = section | literal | emphasis
```

The remainder of this section describes the inline typesetter directives.

FunnelWeb Reference Manual

4.5 Section Directive

The section directive provides a way for the user to structure the program and documentation into a hierarchical tree structure, just as in most large documents. A section construct consists of a case-insensitive identifying letter, which determines the absolute level of the section in the document, and an optional section name, which has exactly the same syntax as a macro name.

```
section   = "@" levelchar [name]
levelchar = "A" | "B" | "C" | "D" | "E" |
            "a" | "b" | "c" | "d" | "e"
```

The section construct is not quite "inline", as it must appear only at the start of a line. However, unlike the "`@i`", "`@p`", and "`@t`" constructs, it does not consume the remainder of the line (although it would be silly to place anything on the same line anyway).

FunnelWeb provides five levels of sections, ranging from the highest level of A (like a LaTeX chapter) to the lowest level of E (like a LaTeX subsubsubsection). FunnelWeb input files need not contain any sections at all, but if they do, the first section must be at level A, and following sections must not skip hierarchical levels (e.g. an @E cannot follow an @C). FunnelWeb generates an error if a level is skipped.

All section *must* have names associated with them, but for convenience, the section name is optional if the section contains one or more macro definitions (i.e. at least one macro definition appears between the section construct in question and the next section construct in the input file.). In this case, the section *inherits* the name of the first macro defined in the section. This feature streamlines the input file, avoiding duplicate name inconsistencies.

Any sequence of printable characters can be used in the section name, even the target typesetter's escape sequence (e.g. in TeX, "\").

The following example demonstrates the section construct.

```
@A@<Life Simulation@>

This is the main simulation module for planet
earth, simulated down to the molecular level. This
is a REALLY big program. I mean really big. I
mean, if you thought the X-Windows source code was
```

```
big, you're in for a shock...

@B We start by looking at the code for six legged
stick insects as they form a good example of a
typical object-oriented animal implementation.

@$@<Six Legged Stick Insects@>@{@-
slsi.creep; slsi.crawl; slsi.creep;@}
```

In the above example, the name for the level A section is provided explicitly, while the name for the level B section will be inherited from the macro name.

FunnelWeb Reference Manual

4.6 Literal Directive

Experience has shown that one of the most common typesetting requirements is that of being able to typeset small program fragments in the middle of the documenting free text. Typically there is a frequent need to refer to program identifiers, and it assists the reader to have such identifiers typeset in the same manner as the program text in the macro definition.

To specify that some text be typeset in tt font, enclose the text in curly brace special sequences as follows.

```
literal = "@{" ordinary_text "@}"
```

As in macro names, section names, and macro bodies, the text contained within the literal construct is protected by FunnelWeb from any non-literal interpretation by the typesetter and the user is free to enclose *any* text covered by the definition ordinary_text. FunnelWeb guarantees that, no matter what the text is, it will be typeset in tt font exactly as it appears. However, the text will be filled and justified into a paragraph as usual.

Here is an example of the use of the construct:

```
@C The @{WOMBAT@} (Waste Of Money, Brains, And
Time) function calls the @{kangaroo@} input
function which has been known to cause keybounce.
This keybounce can be dampened using the
@{wet_sloth@} subsystem.
```

FunnelWeb Reference Manual

4.7 Emphasis Directive

The emphasis directive is very similar to the literal directive except that it causes its argument to be typeset in an emphasised manner (e.g. italics). Like the literal directive, the emphasis directive protects its text argument.

```
emphasise = "@/" ordinary_text "@/"
```

Example:

```
@C What you @/really@/ need, of course, is a
@/great@/, @/big@/, network with packets just
flying @/everywhere@/. This section implements an
interface to such a @/humungeous@/ network.
```

4.8 Macros

The third category of construct appearing at the highest syntactic level in a FunnelWeb input file is the macro definition. A macro definition binds a unique **macro name** to a **macro body** containing an **expression** consisting of text, calls to other macros, and formal parameters. The syntax for a macro definition is as follows:

```
macro = ("@O" | "@$") name [formal_parameter_list]
        ["@Z"] ["@M"] { "@L" } ["==" | "+="]
        "@{" expression "@}"
```

The complexity of the macro definition syntax is mostly to enable the user to attach various attributes to the macro. If the user chooses @O, then the macro cannot be called, but is instead attached to a product file. If the user chooses @$, then the macro is an ordinary macro definition that is not attached to a file. Here are some example macro definitions.

```
@O@<example.txt@>@{This is an @<ugly duckling@>.@}

@$@<ugly duckling@>@M@{swan@}
```

Number Of Invocations Constraint

By default, a non-file macro must be invoked exactly once by one other macro. Macros that aren't are flagged with errors by the FunnelWeb analyser. However, if the user uses the @Z sequence in the macro definition, the macro is then permitted to be invoked zero times, as well as once. Similarly, if the user uses the @M sequence in the macro definition, the macro is permitted to be called many times as well as once. If both @Z and @M are present then the macro is permitted to be invoked zero, one, or many times.

The purpose of enforcing the default "exactly one call" rule is to flag pieces of code that the user may have defined in a macro but not hooked into the rest of the program. Experience shows that this is a common error. Similarly, it can be dangerous to multiply invoke a macro intended to be invoked only once. For example, it may be dangerous to invoke a scrap of non-idempotent initialization code in two different parts of the main function of a program! However, FunnelWeb will not generate an error if a macro without @M is called by another macro that is called more than once.

Additive Macros

If the text string == (or nothing) follows the macro name, the expression that follows is the entire text of the macro body. If the text string += follows the macro name, then more than one such definition is allowed (but not required) in the document and the body of the macro consists of the concatenation of all such expressions in the order in which they occur in the input file. Such a macro is said to be additive and is **additively defined**. Thus a macro body can either be defined in one place using one definition (using ==) or it can be *distributed* throughout the input file in a sequence of one or more macro definitions (using +=). If neither == and += are present, FunnelWeb assumes a default of ==.

Macros attached to product files cannot be additively defined. Additively defined macros can have parameter lists and @Z and @M attributes, but these must be specified only in the first definition of the macro. However, += must appear in each definition.

Library Macros

An ordinary macro definition can have from zero to five @L library level markers. These define the macro definition's library level. A macro having a particular name may be defined up to once at each library level, and each such definition may be multipart (additive). At tangle time, the definition having the lowest library level is used, with all other definitions of the same name being completely ignored. This feature allows the creation of general-purpose include files that contain macro definitions that can be overridden by definitions of the same name in the including file. Similarly, it allows the creation of include files containing macro definitions that override definitions in the including file. How you use the library macro feature is up to you.

In the example below, the ugly duckling macro will be expanded to swan because the swan definition has the lowest library level.

```
@$@<ugly duckling@>@M@L@L@{egg@}
@$@<ugly duckling@>@M@{swan@}
@$@<ugly duckling@>@M@L@{signet@}
```

Note that the library macro facility imposes no requirement on the order of appearance of the various macros of the same name; all that matters is the library level. This means that macros defined in an include file that is included at the *end* of a main file can be overridden by macros appearing earlier in the main file.

FunnelWeb Reference Manual

4.9 Macro Names

Names are used to identify macros and sections. A name consists of a sequence of from zero to 80 printable characters, including the blank character. End of line characters are not permitted in names. Names are case sensitive; two different macros are permitted to have names that differ in case only. Like free text, names are typeset by FunnelWeb and are safe from misinterpretation by the target typesetter. For example, it is quite acceptable to use the macro name @<\medskip@> even if the target typesetter is TeX.

```
name      = "@<" name_text "@>"
name_text = { ordinary_char | text_special }
```

FunnelWeb Reference Manual

4.10 Formal Parameter Lists

FunnelWeb allows macros to have up to nine macro parameters, named @1, @2, ..., @9. If a macro does not have a formal parameter list, it is defined to have no parameters, and an actual parameter list must not appear at the point of call. If a macro has a formal parameter list, it is defined to have one or more parameters, and a corresponding actual parameter must be supplied for each formal parameter, at the point of call.

Because FunnelWeb parameters have predictable names, the only information that a formal parameter list need convey is *how many* parameters a macro has. For this reason a formal parameter list takes the form of the highest numbered formal parameter desired, enclosed in parentheses sequences.

```
formal_parameter_list = "@(" formal_parameter "@)".
formal_parameter = "@1" | "@2" | "@3" | "@4" | "@5" |
                   "@6" | "@7" | "@8" | "@9"
```

FunnelWeb Reference Manual

4.11 Expressions

Expressions are FunnelWeb's most powerful form of expressing a text string. Macro bodies are defined as expressions. Actual parameters consist of expressions.

An expression consists of a sequence of zero or more expression elements. An expression element can be ordinary text, a macro call, or a formal parameter of the macro *definition* in which the formal parameter occurs.

```
expression = { ordinary_text | macro_call |
               formal_parameter }
```

FunnelWeb Reference Manual

4.12 Macro Calls

A macro call consists of a name optionally followed by an actual parameter list. The number of parameters in the actual parameter list must be the same as the number of formal parameters specified in the definition of the macro. If the macro has no formal parameter list, its call must have no actual parameter list.

```
macro_call = name [actual_parameter_list]
actual_parameter_list =
      "@(" actpar { "@," actpar } "@)"
actpar     = expression |
             ( whitespace "@""" expression
               "@""" whitespace )
whitespace = {" " | eol }
```

FunnelWeb allows parameters to be passed directly, or delimited by special double quotes. Each form is useful under different circumstances. Direct specification is useful where the parameters are short and can be all placed on one line. Double quoted parameters allow whitespace on either side (that is not considered part of the parameter) and are useful for laying out rather messy parameters. Here are examples of the two forms.

```
@<Generic Loop@>@(
   @"x:=1;@"  @,
   @"x<=10;@" @,
   @"print "x=%u, x^2=%u",x,x*x;
   x:=x+1;@+@"
@)

@<Colours@>@(red@,green@,blue@,yellow@)
```

The two forms may be mixed within the same parameter list.

Experience has shown that, in most FunnelWeb files, the vast majority of macros have no parameters.

FunnelWeb Reference Manual

4.13 Macro Formal Parameters

Formal parameters can appear in the expressions forming macro bodies in accordance with the syntax rules defined above. A formal parameter expands to the text of the expansion of its corresponding actual parameter. There is nothing preventing a formal parameter being provided as part of an expression that forms an actual parameter. In that happens, the formal parameter is bound to the actual parameter of the calling macro, not the called macro. After the following definitions,

```
@$@<One@>@(@1@)=@{A walrus in @1 is a walrus in vain.@}
@$@<Two@>@(@1@)=@{@<One@>@(S@1n@)@}
```

the call

```
@<Two@>@(pai@)
```

will result in the expansion

```
A walrus in Spain is a walrus in vain.
```

FunnelWeb Reference Manual

4.14 Macros Are Static

In FunnelWeb, the actions of *macro definition* and *macro expansion* occur during two separate phases (parser and tangle) and cannot be interleaved. As a result, the FunnelWeb macro facility is completely static. It is not possible for one macro to define another while the first macro is being expanded; each must be defined statically. It is not possible to define a macro to even assist in the definition of other macros. Because the scanner, parser, analyser, and tangler phases are all invoked sequentially, there is no room for feedback of definitions between different levels (e.g. the user cannot define a macro for the vskip pragma).

This lack of power is fully intentional. By totally excluding the more incomprehensible ways in which a general purpose macro preprocessor can be used, FunnelWeb provides definite guarantees to the reader of its input files:

- FunnelWeb guarantees that a piece of text does not contain a macro call unless it contains the special character followed by < or #.
- FunnelWeb allows calls to be made to macros that are defined later in the input file.

FunnelWeb Reference Manual

5 Analyser

The effect of the parser is to construct a macro table containing a representation of all the macros defined within the document, and a document list which contains a complete representation of the entire document. If there are no error diagnostics (or worse) at the end of the parser run, FunnelWeb invokes the analyser which tests for the following conditions and flags them with errors if they arise.

- No macros defined in the input file.
- No macros connected to output files.
- Call of an undefined macro.
- Call having the wrong number of parameters.
- Call of a macro that is connected to an output file.
- No calls made to a macro without the @Z option.
- More than one call made to a macro without the @M option.
- Directly or indirectly recursively defined macros.
- Unnamed sections that contain no macro definitions.

FunnelWeb performs a static analysis to detect recursion. Unfortunately, the recursion detection algorithm flags all macros that have an infinite expansion rather than just all macros with a recursive definition. If A calls B, and B calls C, and C calls B, then FunnelWeb will flag A as well as B and C. It is hoped that this problem will be fixed in a later version.

Because FunnelWeb does not provide any kind of conditional feature, the prevention of recursion does not represent a curtailment of expressive power.

Macros may be invoked recursively, but may not be recursive. Thus:

```
@! LEGAL   recursive invocation.
@<Sloth@>@(@<Sloth@>@(Walrus@)@)

@! ILLEGAL recursive definition.
@$@<Teapot@>==@{@<Teapot@>@}
```

FunnelWeb Reference Manual

6 Tangle

If the scanner, parser, and analyser have successfully (i.e. with no errors, severe errors, or fatal errors) completed, and the Tangle option (+O) is turned on (it is by default), then the Tangle component of FunnelWeb is invoked to generate the product files specified in the @O macros of the input file.

The operation of Tangle is very simple. Each @O macro is expanded and written to a file of the same name. As there are a finite number of macros, and the analyser guarantees that the macro structure is non-recursive, Tangle is guaranteed to terminate.

Three remaining points are worth mentioning.

- Tangle expands macros using blank indentation unless the user has specified otherwise in an indentation pragma in the input file.
- Tangle keeps track of the length of the lines that it is writing and issues an error if any line of any product file that it generates is longer than the maximum. The maximum is the minimum of a value defaulted or specified in the input file, and the value (if any) provided by the +w command line argument.
- If there is more than one macro definition of the same name, then each such definition must have a different library level (@L). Tangle uses the macro definition that has the lowest library level and ignores the others completely.

Memory Use During Tangling

When FunnelWeb executes, it reads each input file (the main input file and any include files) into memory where they are kept for the duration of the run. This means that there must be room in memory for all of the input files. This approach is necessary to support FunnelWeb's unrestricted forward referencing.

In contrast, there is no requirement that there be enough memory to hold the product files, as these are written to disk sequentially during their expansion. Furthermore, FunnelWeb does not expand the values of actual parameters in memory.

This means that, so long as the input files fit in memory, your product files can be arbitrarily large. You can also pass arbitrary large arguments to FunnelWeb macros.

FunnelWeb Reference Manual

7 Weave

If the scanner, parser, and analyser have successfully (i.e. with no errors, severe errors, or fatal errors) completed, and either or both of the two Weave options (+T, U) is turned on (both are *off* by default), then the Weave component of FunnelWeb is invoked to generate one or two documentation files.

Target Typesetter

All versions of FunnelWeb up to V3.1 could generate only TeX documentation. This used the +t option. FunnelWeb V3.2 can now generate HTML documention using the +u option. You can generate both at once if you like.

Cross Reference Numbering

When FunnelWeb produces its typeset documentation, it *numbers* each section and each macro definition and cross references the macro definitions. The exact scheme used has been carefully thought out. However, as it can be a little confusing to the beginner, it is explained here in full.

The most important thing is that there is *no relation* between the macro numbering and the section numbering. In Knuth's Web there are only section numbers. In FunnelWeb, the numbering of sections and macros is separated.

In FunnelWeb, *sections* are numbered hierarchically in ascending order. For example, the second level-C section of the third level-B section of the first level-A section is numbered "1.3.2". In contrast, *macro definitions* are numbered sequentially in ascending order. For example, the first macro definition is number 1, the second is number 2, and so on. Note that it is *macro definitions* that are numbered, not *macros* . This distinction is necessary because additive macros (i.e. the ones with +=) can be defined by a collection of partial definitions scattered throughout the input file. A single additive macro may be defined in definitions 5, 67, 128, and 153.

FunnelWeb Reference Manual

8 FunnelWeb Shell

FunnelWeb Reference Manual

8.1 Introduction

One of the goals of FunnelWeb is that it must be extremely portable, and a significant effort has gone into achieving this.

An equally important goal was that of correctness and reliability. To this end, it was determined that a large automated suite of test programs be prepared to assist in regression testing. Preparing the test suite was tedious, but achievable. Automating it portably was more difficult.

The difficulty faced was that if FunnelWeb was implemented in the form of a utility that could be invoked from the operating system command language, the only way to set up regression testing was in the command language of the operating system of the target machine (shellscripts for UNIX, DCL for OpenVMS, batch files for MSDOS, and *nothing* on the Macintosh). The huge variation in these command languages led to the conclusion that either the automation of regression testing would have to be rewritten on each target machine, or a small command language would have to be created within FunnelWeb. In the end, the twin goals of portability and regression testing were considered so important that a small command shell was constructed inside FunnelWeb. This is called the **FunnelWeb command shell**, or just "the shell" for short.

By default, when FunnelWeb is invoked, it does not enter its shell. If just given the name of an input file, it will simple process the input file in the normal manner and then terminate. To instruct FunnelWeb to invoke its shell, the +K or +X command line option must be specified when FunnelWeb is invoked from the operating system. It is also invoked upon startup if the file fwinit.fws exists.

Most FunnelWeb users will never need to use the shell and need not even know about it. There are four main uses of the shell:

1. As a tool to support automated regression testing.
2. As a development tool on machines that do not have a built in shell (e.g. the Macintosh). The shell can be used to process whole groups of files automatically.
3. As a convenience. A user working on a multi-tasking,

multi-window workstation may wish to keep an interactive session of FunnelWeb going in one window rather than having to run up the utility each time it is required.

4. As a convenient vehicle for enclosing utilities. The FunnelWeb shell contains useful general purpose commands such as the differences command diff.

FunnelWeb Reference Manual

8.2 Return Statuses

The hierarchy of diagnostics is also used in the shell commands. Each shell command returns a status which can affect further processing.

Success status is the normal command return status.

Warning status is returned if some minor problem arose with the execution of the command.

Error status is returned if a significant problem arises during the execution of the command. However, unlike a severe error, it does *not* cause termination of the enclosing shellscript.

Severe error status is returned if a problem arises during the execution of the command that prevents the command from delivering on its "promise". A severe error causes FunnelWeb to abort the script (and any stacked scripts) to the interactive level. (However, the tolerate command allows this to be temporarily overridden).

Fatal error status is returned if a problem arises that is so serious that execution of FunnelWeb cannot continue. A fatal error causes FunnelWeb to abort to the operating system level.

Assertion error status is never returned. If an assertion error occurs, FunnelWeb bombs out ungracefully to the operating system. Assertion errors should never happen. If they do, then there is a bug in FunnelWeb.

To be precise, the status returned by each command is a vector of numbers being the number of each of the different kinds of diagnostic generated by the command. Usually only one kind of diagnostic is generated. However, the fw command and a few of the other commands can generate more than one kind of diagnostic. These status vectors are summed internally where they may later be accessed using the status command. However, the current diagnostic state evaporates as soon as the next command is encountered.

FunnelWeb Reference Manual

8.3 Command Line Length

The maximum length of a shell command line is guaranteed to be at least 300 characters.

FunnelWeb Reference Manual

8.4 String Substitution

Most command shells provide some form of string substitution so as to provide some degree of parameterization. The FunnelWeb shell provides 36 different string variables named $0..$9 and $A..$Z (case insensitive). Each variable can hold a string containing any sequence of printable characters and can be as long as a command line.

The define command allows the user to assign a value to these variables. The define command takes two arguments. The first is the digit or letter of the variable to be defined. The second is a double quote delimited string being the string value to be assigned to the variable. If you want to include a double quote character within the string, you don't need to double it.

Examples:

```
define 3 "/root/usr/dave/workdir/fwdir/testdir"
define M "/user/local/rubbish/bin/fw"
define Q "You don't need to double" double quotes"
```

Only the identifying character of the variable being assigned is used in the definition. This syntax is a simple way of preventing the variable from being substituted before it has a chance to be defined!

The following points clean up the remaining semantic details:

- There is only one set of variables and they are global to all shellscripts. There are *no local variables* .
- When a shellscript is invoked using the execute command, the substitution variables 0 through 9 are affected. See the EXECUTE command for more details.
- If you want to include a dollar sign character in a command use "`$$`".
- FunnelWeb also defines "`$/`" which translates to the character that separates directory and file name fields in file names on the host machine. For example: Sun="`/`", Vax="`]`", Mac="`:`", PC="`\`".
- Substitution is not performed recursively.

FunnelWeb Reference Manual

8.5 How a Command Line is Processed

When FunnelWeb reads in a command line (from the console or a script file), it processes it in the following sequence:

1. The command line is checked for non-printable characters. If there are any, they are flagged with a severe error.
2. All dollar string substitution variables in the command line are replaced by their corresponding string. The command line is processed from left to right. Substitutions are performed non recursively.
3. At this point, if the line is empty, or consists entirely of blanks, it is ignored and the interpreter moves to the next line.
4. A severe error is generated if the line at this stage begins with a blank.
5. If the first character of the line is "!", the line is a comment line and is ignored.
6. The run of non-blanks commencing at the start of the line is compared case-insensitively to each of the legal command verbs. If the command is illegal, a severe error is generated, otherwise the command is processed.

FunnelWeb Reference Manual

8.6 Options

The FunnelWeb shell maintains three sets of command line options.

1. The set of options resulting from applying the operating system level command line arguments to the default option settings.
2. A set of shell options that prevail during the shell invocation.
3. The set of option values active during a particular invocation of FunnelWeb proper.

When FunnelWeb is invoked from the operating system with just +F, only the first of these three sets comes into existence. If the user invokes the FunnelWeb shell, the shell options come into existence and are initialized with the value of the first set. These shell options are used as the default for all subsequent fw commands. However, they can be altered using the script command set. If a fw command executed in a shell contains additional command line options, these override the shell options for that run, but do not change the shell options. An example follows:

```
$ fw +k +t            ! Original invocation from OS.
                      ! Options default with "+t".
FunnelWeb>fw sloth    ! Equivalent to fw sloth +t.
FunnelWeb>set -l      ! Change the l shell option.
FunnelWeb>fw sloth +q ! Equiv to fw sloth +t -l +q.
FunnelWeb>fw sloth    ! Equiv to fw sloth +t -l.
```

The existence of the shell option set means that the user can set up a set of defaults to be applied to all fw commands issued within the shell.

FunnelWeb Reference Manual

9 Shell Commands

FunnelWeb Reference Manual

9.1 Introduction

This section describes each of the FunnelWeb shell commands. The syntax is:

```
shell_command = absent   | codify  | compare     |
                define   | diff    | diffsummary |
                diffzero | eneo    | execute     |
                exists   | fixeols | help        |
                here     | quit    | set         |
                show     | skipto  | status      |
                tolerate | trace   | write       |
                writeu
s = { " " }+
```

As a rule, FunnelWeb shell commands return severe status if their arguments are syntactically incorrect or if they are unable to successfully operate on argument files.

FunnelWeb Reference Manual

9.2 Absent

The absent command performs no action except to return a status. If the file specified in its argument doesn't exist it returns success status, otherwise it returns severe status.

```
Syntax : absent = "absent" s filename
Example: absent result.out
```

This command is useful in regression testing for making sure that FunnelWeb *hasn't* produced a particular output file.

FunnelWeb Reference Manual

9.3 Codify

The codify command takes two arguments: an input file and an output file. It reads each line of the input file and writes a corresponding line to the output file. The corresponding line consists of a C macro call containing a string containing the input line. The command converts all backslashes in input lines to double backslashes so as to avoid unwanted interpretations by the C compiler. It also converts double quotes in the line to backslashed double quotes.

```
Syntax : codify = "codify" s filename s filename
Example: codify header.tex header.c
```

The following example demonstrates the transformation.

```
Input  Line:
 \def\par{\leavevmode\endgraf}% A "hack".
Output Line:
 WX("\\def\\par{\\leavevmode\\endgraf}% A \"hack\".");
```

The codify command was introduced to assist in the development of FunnelWeb. It is used to convert longish text files into C code to write them out. The C code is then included within the FunnelWeb C program. For example, the set of TeX definitions that appears at the top of every documentation file was codified and inserted into the FunnelWeb code so that FunnelWeb would not have to look for a file containing the definitions at run time.

FunnelWeb Reference Manual

9.4 Compare

The compare command takes two filename arguments and performs a binary comparison of the two files. If the files are identical, success status is returned. If they are different, severe status is returned. No information about the manner in which the files differ is conveyed.

```
Syntax : compare = "compare" s filename s filename
Example: compare result.txt answer.txt
```

The compare command was created as the main checking mechanism for regression testing. However, its binary output was soon found to be unworkable and the more sophisticated diff command was added so that the actual differences between the files could be examined.

FunnelWeb Reference Manual

9.5 Define

The define command assigns a value to a shell string substitution variable. The define command takes two arguments. The first is the digit or letter of the variable to be defined. The second is a double quote delimited string being the string value to be assigned to the variable. If you want to include a double quote character within the string, you don't need to double it.

```
Syntax  :
   define = "define" s letter s """" text """"
Examples:
   define 3 "/usr/usrs/thisuser/workdir/fwdir/testdir"
   define M "/user/local/rubbish/bin/fw"
   define Q "You don't need to double" double quotes"
```

The command interpreter expands the command line before it executes the define command. This means that you can define string substitution variables in terms of each other with static binding.

The define command was introduced to allow the parameterization of the directories involved in regression testing.

See the section on string substitution for more details.

FunnelWeb Reference Manual

9.6 Diff

The diff command reads in two text files and *appends* a report to a log file containing a list of the differences between the two input files. If the log file does not already exist, an empty one is created first.

```
Syntax  :
   diff = "diff" s filename s filename s
                   filename s ["ABORT"]
Examples:
   diff result.tex answer.tex diff.log
   diff $Otest23.out $Atest23.out $Ldiff.log ABORT
```

The diff command performs a full line-based differences operation. It will identify different sections in a file, even if they are of differing length.

The implementation of the diff command is quite complicated. To be sure that it is at least getting its same/different proclamation right, the diff command performs a binary comparison as an extra check.

The following points describe the rules for determining the result status.

1. diff aborts with a severe error if the log file cannot be opened or created for appending.
2. An ordinary error is generated if either or both of the input files cannot be opened.
3. If, at the end of the run, the two input files have not been proven to be identical, and the ABORT keyword is present, diff returns severe status.
4. diff returns success status if none of the above conditions (or similar conditions) occur, even if the two files are different.

The diff command *appends* its differences report rather than merely writing it. This allows a regression test script to perform a series of regression tests and produce a report for the user.

The diff command was added to the shell after it had become apparent that the simpler compare command was not yielding enough information. Whereas early on, regression testing was treated mainly as a tool to ensure that FunnelWeb was being ported to other machines correctly, it began to place an increasing role during development in identifying the effects of changes made to the code. The diff command supports this application of regression testing by pinpointing the differences between nearly-identical text files.

FunnelWeb Reference Manual

9.7 Diffsummary

The diffsummary command writes a short report to the console giving the number of difference operations that have taken place and how many of the pairs of files compared were identical. Counting starts at the most recent execution of a diffzero command, or if there has been none, when FunnelWeb started up.

```
Syntax  : diffsummary = "diffsummary"
Examples: diffsummary
```

The diffsummary command was added so as to allow regression testing scripts to display a summary of the results of the test. If the summary indicates that no pair of files differed, then there is no need to look in the diff log file.

FunnelWeb Reference Manual

9.8 Diffzero

The diffzero command zeros the different summary counters used by the diff and diffsummary commands.

```
Syntax  : diffzero = "diffzero"
Examples: diffzero
```

The diffzero command was added so as to allow regression testing shellscripts to zero their differences counters at the start of a run. This allows testers to invoke the same regression testing script twice in one interactive session without receiving an inflated differences summary.

FunnelWeb Reference Manual

9.9 Eneo

The eneo command takes one filename argument. If the file does not exist, no action is taken. If the file does exist, it is deleted. In both cases success status is returned. However, if the file exists and cannot be deleted, eneo returns severe status.

```
Syntax  : eneo = "eneo" s filename
Examples: eneo result.out
```

The eneo command was added so as to allow regression testing scripts to ensure that existing output files were not present before proceeding with a test run. If FunnelWeb were to fail to generate an output file, it would be extremely undesirable for the old version to be used.

ENEO stands for **E**stablish the **N**on **E**xistence **O**f. Most operating systems provide a command to delete files. Typically these commands are verbs such as "delete", "remove", and "kill". As a consequence, the designers of delete commands usually consider the command to have failed if it fails to find the file to be deleted. However, in scripts, the delete command is very commonly used to *establish the non-existence of* one or more files. Typically, a script is starting up and needs to clear the air before getting started. If the files are there, they should be deleted; if they are not, then that's OK too. (Note: As far as I know, the eneo command is original).

FunnelWeb Reference Manual

9.10 Execute

The execute command causes a specified text file to be executed as a FunnelWeb shellscript. The first argument is the name of the script file. The remaining arguments are assigned to the substitution variables $1, $2, ..., $9. Substitution variables in the range $1 to $9 that do not correspond to an argument are set to the empty string "". $0 is set to the empty string regardless. The execute command can be used recursively, allowing shell scripts to invoke each other. A file extension default of "`.fws`" (FunnelWeb Script) applies to script files.

```
Syntax  :
   execute = "execute" s filename {argument_string}
Examples:
   execute megatest.fws /usr/users/ross/fwtest !
   execute sloth
```

The first example above will result in the following substitution variable assignments.

```
$0 = ""
$1 = "/usr/users/ross/fwtest"
$2 = "!"
$3 = ""
...
$9 = ""
```

It should be stressed that there are no local variables in the FunnelWeb command language; the variables above are globally modified.

The execute command was added to allow the creation of sub-scripts to test FunnelWeb in particular ways.

FunnelWeb Reference Manual

9.11 Exists

The exists command performs no action except to return a status. If the file specified in its argument exists it returns success status, otherwise it returns severe status.

```
Syntax : exists = "exists" s filename
Example: exists test6.fw
```

This command is useful in regression testing for ensuring that FunnelWeb has produced a particular output file.

FunnelWeb Reference Manual

9.12 Fixeols

The fixeols command takes two filename arguments: an input file and an output file. It reads in the input file and writes it to the output file changing all the end of line control character sequences to the local format. It can also take one filename argument, in which case it replaces the target file with its transformation.

```
Syntax  : fixeols = "fixeols" s filename [s filename]
Examples: fixeols imported.hak result.kln
          fixeols sloth.dat
```

The fixeols command works by parsing the input file into alternating runs of printable characters (ASCII 20 to ASCII 126) and runs of non-printable characters (all the others). It then parses each run of non-printable characters from left to right into subruns of non-printables not containing the same character twice. It then replaces each subrun with a native EOL. (Note: A native EOL can be inserted into a text file in a portable manner simply by writing "`\n`" to the text output stream). For example, if a native EOL is X, and ABCD are non-printable characters, and the file to be converted is

```
thisABisABCDanABABexampleABCCCof the conversion.
```

then fixeols would produce

```
thisXisXanXXexampleXXXof the conversion.
```

The fixeols command was devised to solve the problem created sometimes when text files are moved from one machine to another (e.g. with the Kermit program) using a binary transfer mode rather than a text transfer mode. If such a transfer is made, and the text file line termination conventions differ on the two machines, one can wind up with a set of text files with improperly terminated lines. This can cause problems on a number of fronts, but in particular affects regression testing which relies heavily on exact comparisons between files. The fixeols command provides a solution to this problem by providing a portable way to "purify" text files whose end of lines have become incorrect. The regression testing scripts all apply fixeols to their input and output files before each test.

FunnelWeb Reference Manual

9.13 Fw

The fw command allows FunnelWeb proper to be invoked from a shell script. The syntax is almost identical to the syntax with which FunnelWeb is invoked from the operating system.

```
Syntax  : fw = "fw" s ordinary_funnelweb_command_line
Examples: fw sloth +t +d
          fw -l walrus
```

Some important points about this fw command are:

- Options are inherited from the default shell options.
- The F (input file option) must be turned on.
- The K, H, and X options must be turned off.
- The J option must be turned off.
- The options specified in a fw command do not affect the default shell options.
- This command performs no action in the OpenVMS version of FunnelWeb.

FunnelWeb Reference Manual

9.14 Help

The help command provides online help from within the FunnelWeb shell. It provides access to all of the same messages that the +H command line option does.

```
Syntax  : help = "help" [s help_message_name]
Examples: help
          help commands
```

If no message name is given, the default message is displayed. It contains a list of the other help messages and their names. The actual messages themselves are not listed here.

FunnelWeb Reference Manual

9.15 Here

The here command acts as a target for the skipto command. When the shell interpreter encounters a skipto command, it ignores all the following commands until it encounters a here command.

```
Syntax : here = "here"
Example: here
```

The skipto/here mechanism was created to allow groups of regression tests to be skipped during debugging without having to comment them out.

FunnelWeb Reference Manual

9.16 Quit

The quit command terminates FunnelWeb immediately and returns control to the operating system. This applies regardless of the depth of the script being executed.

```
Syntax : quit = "quit"
Example: quit
```

FunnelWeb Reference Manual

9.17 Set

The set command modifies the default shell options. For example, set +t sets the +t option for all subsequent FunnelWeb runs within the shell until another set command sets -t.

```
Syntax:
   set = "set" s ordinary_funnelweb_command_line
Examples:
   set sloth +t +d
   set -lwalrus
```

The restrictions on the set command are identical to those on the fw command except that, in addition, the +F option cannot be turned on in the set command.

The set command is useful for setting option defaults before a long run of regression tests. It could also be useful to set default options in a FunnelWeb shell kept by a user in a workstation window.

FunnelWeb Reference Manual

9.18 Show

The show command displays the current default shell options. These options are the options that subsequent fw commands will inherit.

```
Syntax : show = "show"
Example: show
```

FunnelWeb Reference Manual

9.19 Skipto

The skipto command causes the shell to ignore all subsequent commands until a here command is encountered.

```
Syntax  : skipto = "skipto"
Examples: skipto
```

The skipto/here mechanism was created to allow groups of regression tests to be skipped during debugging without having to comment them out. It is like a cut price goto. For example, supposing that there were eight tests and that you had debugged the first five. You might want to skip the first five tests so that you can concentrate on the next three. The following code shows how this can be done.

```
skipto
execute test infile1
execute test infile2
execute test infile3
execute test infile4
execute test infile5
here
execute test infile6
execute test infile7
execute test infile8
```

It should be stressed that FunnelWeb performs full command line processing including the dollar substitutions before testing the line to see if it is here. This can lead to non-obvious problems. For example.

```
skipto
! Test the Parser
! ---------------
define X "execute parsertest.fws"
$X infile1
$X infile2
$X infile3
$X infile4
$X infile5
here
```

The above looks correct, but, because the define command isn't executed (and $X is not defined) the subsequent $X lines result in a leading blanks error. The problem can be corrected by defining $X before the skipto command.

FunnelWeb Reference Manual

9.20 Status

The status command takes two forms. In its first form in which no arguments are given, it writes out the number of warnings, errors and severe errors that 1) were generated by the previous command and 2) have been generated during the entire shell invocation. In its second form it takes from one to three arguments each of which specifies a diagnostic severity and a number. The status command compares each of these numbers with the number of that diagnostic generated by the previous command and generates a severe error if they differ.

```
Syntax  : status = "status" {s ("w"|"e"|"s") num}0..3
Examples: status
          status w1 e5 s1
          status w4
          status s1 e2
```

The status command was introduced to test the status results of commands during their debugging. It is also useful for checking to see that the right number of diagnostics have been generated at particular points in test scripts.

FunnelWeb Reference Manual

9.21 Tolerate

The tolerate command instructs the shell not to abort processing of the script if the next command generates one or more warnings, errors, or severe errors. For the purposes of this command, a blank line counts as a command, so be sure to place the tolerate command immediately above the command about which you wish to be tolerant.

```
Syntax : tolerate = "tolerate"
Example: tolerate
```

The tolerate command was introduced to allow FunnelWeb (i.e. the fw command) to be tested in a script under conditions which would normally cause it to abort the script.

FunnelWeb Reference Manual

9.22 Trace

The trace command turns on or off command tracing during script execution. By default, tracing is turned off.

```
Syntax  : trace = "trace" [s ("on" | "off")]
Examples: trace on
          trace off
```

The trace command was introduced to assist in the debugging of regression test scripts.

FunnelWeb Reference Manual

9.23 Write

The write command accepts a double-quoted argument and writes it followed by an EOL to the console (standard output). There is no need to double any double quotes occurring within the string.

```
Syntax:
   write = "write" s string
Examples:
   write "Now about to start the next test."
   write "No need to " double enclosed double quotes."
```

The write command was added so as to allow regression testing scripts to inform the user of their progress.

FunnelWeb Reference Manual

9.24 Writeu

The writeu command is identical to the write command except that it underlines the text on an additional following output line.

```
Syntax  : writeu = "writeu" s string
Examples: writeu "Test 6"
```

FunnelWeb Reference Manual

10 Glossary

Analyser: A component of the FunnelWeb program that checks the macro table created by the parser for errors. For example, the analyser checks to see if any macro without a @Z has not been called.

Argument: A string delimited by blanks appearing on the FunnelWeb command line. Arguments are used to control options.

Directive: A FunnelWeb special sequence or cooperating group of special sequences that do not form part of a macro definition. A directive can take the form of a pragma.

Documentation: Descriptive text.

Documentation file: An output file, produced by the Weave component of FunnelWeb, that contains typesetter commands. When fed into the appropriate typesetter program, the result is a typeset image of the input file.

Free text: The text in an input file that remains if one were to remove macro definitions and directives.

FunnelWeb: This word has a number of different meanings all pertaining to the FunnelWeb system of programming. 1) The entire system of programming as in "Maybe FunnelWeb can help." 2) The computer program that implements the system as in "Run it through FunnelWeb and see what comes out." 3) The language implemented by the FunnelWeb program as in "I wrote the program in FunnelWeb." or "I wrote the program in Ada using FunnelWeb.".

FunnelWeb file: A file whose contents are written in the FunnelWeb language.

FunnelWeb language: The language in which FunnelWeb input files are written.

FunnelWeb proper: Usually, when FunnelWeb is invoked, it processes a single input file and then terminates. However, it also has a command language mode in which it is possible to invoke "FunnelWeb" many times. This leads to confusion between "FunnelWeb" the outer program and "FunnelWeb" the inner program. To avoid this confusion, the inner FunnelWeb is sometimes referred to as "FunnelWeb proper".

FW: An abbreviation for "FunnelWeb" that is used wherever appropriate.

Include file: A file read in by FunnelWeb as the result of an include pragma (@i filename).

Input file: Any file read in by FunnelWeb. The phrase "the input file" refers to the root input file (specified using the +F option).

Journal file: An output file containing a copy of the output sent to the user's console during an invocation of FunnelWeb. In other systems, this file is sometimes called a "log file".

Listing file: An output file summarizing the result of processing an input file.

Macro: A binding of a name to a string.

Macro definition: A construct appearing in a FunnelWeb file that binds a name to a text string. A FunnelWeb file consists of a series of macro definitions surrounded by documentary text.

Mapper: A component of the FunnelWeb program that reads in the input file and creates a copy of it in memory.

Option: An parameter internal to the FunnelWeb program which can be controlled by command line arguments or pragmas.

Output file: Any file written by FunnelWeb. This includes listing, journal, product, and documentation files. (Warning: During most of FunnelWeb's development the term "output file" was also used to refer to what are now called "product files". This turned out to be extremely confusing and so the term "product file" was invented to distinguish the generic from the specific. However, as this was a late modification, you may find some occurrences of the old use of "output file".).

Parser: A component of the FunnelWeb program that processes the token list generated by the scanner and produces a macro table and a document list. The parser mainly analyses the input file at the syntactic level, but also does some lightweight semantic checking too.

Pragma: Single-line directives that appears in FunnelWeb files. Pragmas control everything from maximum input line length to typesetter dependence. A pragma line starts with "`@p`"

Printed documentation: Sheets of paper resulting from actually typesetting and printing a documentation file.

Product file: An output file, generated by the Tangle component of FunnelWeb, that contains the expansion of the macros in the input file. Note: Other names considered for this were: generated file, expanded file, result file, program file, and tangle file.

Scanner: A component of the FunnelWeb program that scans a copy of the input file in memory and generates a line list and a token list to be fed to the parser. The scanner processes the input at the lexical level.

Script: A file containing FunnelWeb shell commands.

Shell: A command language interpreter built into the FunnelWeb program. The interpreter allows the user to invoke FunnelWeb proper many times during a single invocation of the FunnelWeb program.

Special character: A distinguished character in a FunnelWeb input file that introduces a special sequence. By default the special character is "`@`". However, it can be changed using the "`@=`" special sequence.

Special sequence: A special sequence is a construct introduced by the special character. Special sequences are used to define a structure in a FunnelWeb input file that exists at a higher level to the surrounding text. A FunnelWeb input file may be considered to be a sequence of text and special sequences.

Tangle: This is the name for the component of FunnelWeb that generates one or more product files containing the expansion of macros in the input file.

Typesetting directive: A FunnelWeb directive whose sole effect is to modify the way in which the input file is represented in the documentation file.

Weave: This is the name for the component of FunnelWeb that generates a documentation file containing typesetting commands representing the input file.

FunnelWeb Reference Manual

11 References

[ANSI] Australian Standard AS 3955-1991, "Programming Languages --- C", (ISBN: 0-7262-6970-0), 12 July 1991. Identical to: International Standard ISO/IEC 9899: 1990 Programming Languages --- C.

[ANZE] "Australia, New Zealand Encyclopedia", Entry: "Funnel-web spiders", Vol 7, pp. 564--565, Bay Books, Sydney, (ISBN: 85835--127--7), 1975.

[BSI82] British Standards Institute, "Specification for Computer Programming Language Pascal", Publication BS6192:1982, British Standards Institute, P.O. Box 372, Milton Keynes, MK146LO, 1982.

[Gries81] Gries D., "The Science of Programming", Springer-Verlag, (ISBN: 0-387-90641-X), 1981.

[Humphries91] Humphries B, "Neglected Poems and Other Creatures", Angus and Robertson, Sydney, (ISBN: 0-207-17212-9), 1991.

[Kernighan88] Kernighan B.W., Ritchie D.M., "The C Programming Language", (second edition,"ANSI C"), Prentice Hall, (ISBN: 0-13-110362-8), 1988.

[Knuth83] Knuth D.E., "The WEB System of Structured Documentation", (Web User Manual, Version 2.5, November, 1983), Stanford University, 1983.

[Knuth84] Knuth D.E., "The TeXbook", Addison-Wesley, (ISBN: 0-201-13448-9), 1984.

[Knuth84] Knuth D.E., "Literate Programming", *The Computer Journal* , Vol. 27, No. 2, pp. 97-111, 1984. (Reference copied from SIGPLAN 26(1) p.16). Note: The author of this manual has not yet obtained this paper.

[Lamport86] Lamport L., "LaTeX: A Document Preparation System", Addison-Wesley, (ISBN: 0-201-15790-X), 1986.

[Rosovsky90] Rosovsky H., "The University: An Owner's Manual", W.W.Norton & Company, Inc., (ISBN: 0-393-02782-1), 1990.

[Smith91] Smith L.M.C., "An Annotated Bibliography of

Literate Programming", ACM SIGPLAN Notices, Vol. 26, No. 1, January 1991.

[Strunk79] Strunk W., White E.B., "The Elements of Style", Third Edition, MacMillan Publishing Company, New York, (ISBN: 0-02-418200-1), 1979.

[USDOD83] "The Programming Language Ada Reference Manual", American National Standards Institute Inc, ANSI/MIL-STD-1815A-1983, 1983.

FunnelWeb Reference Manual

Search FunnelWeb Documentation

Information about FunnelWeb is divided into the main FunnelWeb web and the Tutorial, Reference, and Developer manual webs. Choose a combination of manuals to search, and enter one or more keywords.

- ☐ FunnelWeb Main Web (General information)
- ☑ FunnelWeb Reference Manual (Official definition)
- ☐ FunnelWeb Tutorial Manual (Tutorial and hints)
- ☐ FunnelWeb Developer Manual (How to compile)

* Enter one or more words or word prefixes separated by spaces.
* Matching is case insensitive.
* Finds all pages containing at least one word.
* Add the word AND to find pages containing all the words.
* Searching will not work in offline copies of this web.

Search

FunnelWeb Reference Manual

Copyright and Credits

Copyright Of The FunnelWeb Program

Copyright Of The FunnelWeb Webs

Trademarks

Macintosh is a trademark of Apple Computer
MS-DOS is a trademark of Microsoft.
Rocksoft is a registered trademark of Rocksoft Pty Ltd, Australia.
Unix is a registered trademark of AT&T.
VAX and **OpenVMS** are trademarks of Digital Equipment Corporation.

Questions

Please email me if you have any other questions about the intellectual property aspects of FunnelWeb.

Credits

FunnelWeb was conceived, designed and implemented by Ross Williams in 1986, 1992, and 1999. The FunnelWeb webs were originally written by Ross Williams in 1992 in the form of a printed manual, and converted by him to webs in April 1999.

FunnelWeb was the main tool used to create these FunnelWeb webs. Each web was written as a single FunnelWeb .fw file which, when processed by FunnelWeb, generates all the .shtml files.

I would like to thank a number of people who assisted me (Ross Williams) during the creation of FunnelWeb.

Many thanks to David Hulse for translating the original version of FunnelWeb (FunnelWeb V1) from Ada into C (FunnelWeb V2) and getting it to work on Unix and a PC. The C code written by David (FunnelWeb V2) formed the basis of FunnelWeb V3.

Thanks go to Simon Hackett of Internode Systems for the use of his Sun, Mac, and PC, for assistance in porting FunnelWeb to the Sun and PC, and for helpful discussions.

Thanks go to Jeremy Begg of VSM Software Services for the use of his VAX, and for assistance with the VMS-specific code.

Thanks to Barry Dwyer and Roger Brissenden for trying out FunnelWeb Version 1 in 1987 and providing valuable feedback.

Thanks to Donald Knuth for establishing the idea of literate programming in the first place.
