Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

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

THIS TUTORIAL MANUAL provides a friendly introduction to the FunnelWeb literate programming preprocessor

This Tutorial Manual does not provide a definitive description of FunnelWeb, so if you have a specific technical question, you should refer to the FunnelWeb Reference Manual, which is definitive. To perform a keyword search of the Reference Manual and/or this Tutorial manual, click on SEARCH in the margin.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

1 Introduction

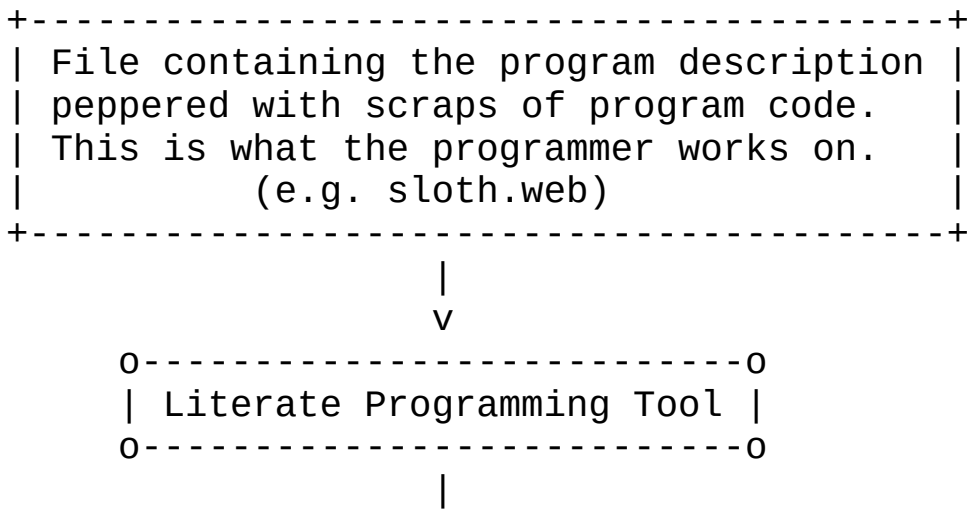
1.1 What Is Literate Programming?

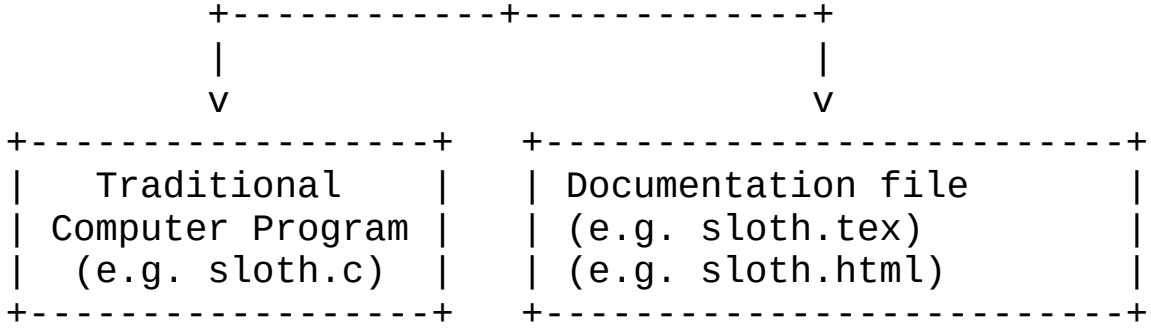
A traditional computer program consists of a text file containing program code. Scattered in amongst the program code are comments which describe the various parts of the code.

In **literate programming** the emphasis is reversed. Instead of writing code containing documentation, the literate programmer writes documentation containing code. No longer does the English commentary injected into a program have to be hidden in comment delimiters at the top of the file, or under procedure headings, or at the end of lines. Instead, it is wrenched into the daylight and made the main focus. The "program" then becomes primarily a document directed at humans, with the code being herded between "code delimiters" from where it can be extracted and shuffled out sideways to the language system by literate programming tools.

The effect of this simple shift of emphasis can be so profound as to change one's whole approach to programming. Under the literate programming paradigm, the central activity of programming becomes that of conveying meaning to other intelligent beings rather than merely convincing the computer to behave in a particular way. It is the difference between performing and exposing a magic trick.

In order to program in a literate style, particular tools are required. The traditional approach (used in the FunnelWeb system) is to have some sort of text-file-in/text-file-out utility that reads a literate program (containing a program commentary peppered with scraps of program text) and writes out a file containing all the program code and a file containing typesetter commands representing the entire input document, documentation, code, and all. See the diagram below.





Traditional architecture of literate programming tools.

Literate programming tools could be organized in a number of ways. However, to fit in with current file and command line based environments, most tools conform to the traditional architecture shown here in which the user feeds in a file containing a literate program, and the literate programming utility generates program files and a documentation file.

Given the coming age of hypertext systems, this is probably not the best approach. However, it does mesh beautifully with current text files and command line interfaces, the expectation of linear presentations in the documents we read, and the particular requirements of current programming languages and typesetting systems. It is certainly not a bad approach.

With this structure in place, the literate programming system can provide far more than just a reversal of the priority of comments and code. In its full blown form, a good literate programming facility can provide total support for the essential thrust of literate programming, which is that computer programs should be written more for the human reader than for the compiler. In particular, a literate programming system can provide:

Re-ordering of code: Programming languages often force the programmer to give the various parts of a computer program in a particular order. For example, the Pascal programming language**[BSI82]** imposes the ordering: constants, types, variables, procedures, code. Pascal also requires that procedures appear in an order consistent with the partial ordering imposed by the static call graph (but forward declarations allow this to be bypassed). In contrast, the literate style requires that the programmer be free to present the computer program in any order whatsoever. The facility to do this is implemented in literate programming tools by providing text *macros* that can be defined and used in any order.

Typeset code and documentation: Traditionally program listings are dull affairs consisting of pages of fan-form paper imprinted with meandering coastlines of structured text in a boring font. In contrast, literate programming systems are capable of producing documentation that is superior in two ways. First, because most of the documentation text is fed straight to the typesetter, the programmer can make use of all the power of the underlying

typesetter, resulting in documentation that has the same presentation as an ordinary typeset document. Second, because the literate programming utility sees all the code, it can use its knowledge of the programming language and the features of the typesetting language to typeset the program code as if it were appearing in a technical journal. It is the difference between:

```
while sloth<walrus loop
   sloth:=sloth+1;
end loop
```

and

while *sloth* <*walrus* **loop**
    sloth =*sloth* +1;
end loop

Unfortunately, while FunnelWeb provides full typesetting of the documentation, it typesets all of its code in the style of the first of these two examples. To typeset in the style of the second requires knowledge of the programming language, and the current version of FunnelWeb is programming language independent. At a later stage, it is possible that FunnelWeb will be modified to read in a file containing information about the target programming language to be used to assist in typesetting the code properly.

Cross referencing: Because the literate tool sees all the code and documentation, it is able to generate extensive cross referencing information in the typeset documentation. This makes the printed program document more easy to navigate and partially compensates for the lack of an automatic searching facility when reading printed documentation.

In the end, the details don't matter. The most significant benefit that literate programming offers is *its capacity to transform the state of mind of the programmer* . It is well known that the act of explaining something can transform one's understanding of it. This is one of the justifications behind the powerful combination of research and teaching in universities **[Rosovsky90]**. Similarly, by constantly explaining the unfolding program code in English to an imaginary reader, the programmer transforms his perception of the code, laying it open, prone, to the critical eye.

The result of this exposure is a higher quality of programming. When exposed to the harsh light of the literate eye, bugs crawl out, special cases vanish, and sloppy code evaporates. As a rule, literate programs take longer to write than ordinary programs, but the total development time is the same or less because the time taken to write and document the program carefully is compensated for by a reduced debugging and maintenance time. Thus literate programming does not merely assist in the preparation of documentation, but also makes significant contributes to the process of programming itself. In practice this has turned out

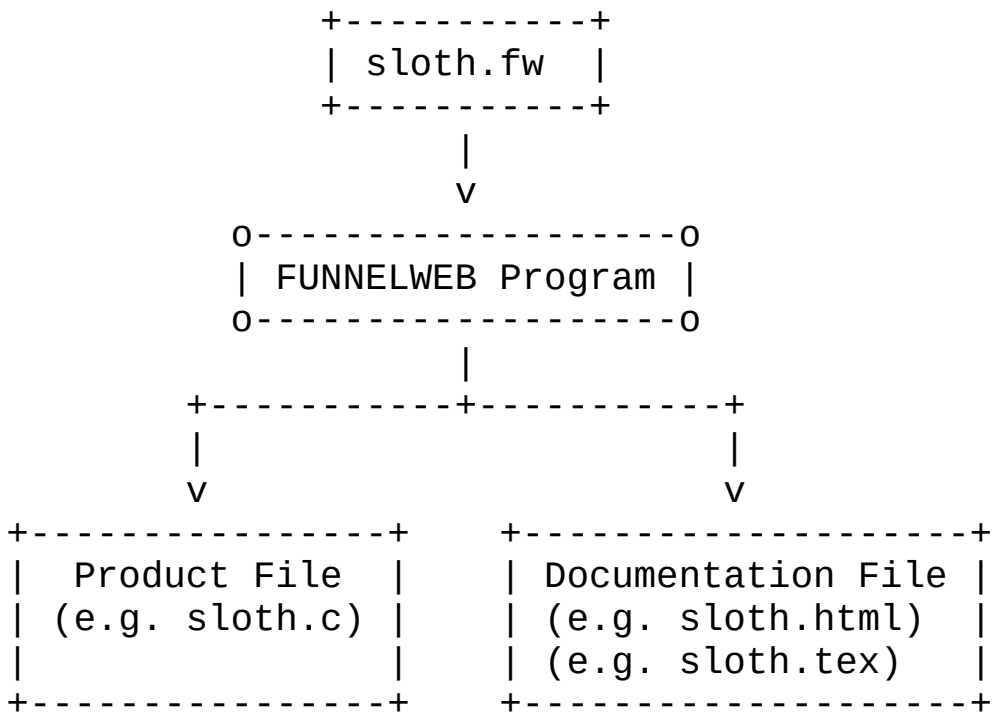
to be a contribution far more important than the mere capacity to produce typeset documentation.

For more information on literate programming, the reader is directed to Knuth's early founding work **[Knuth83]** and **[Knuth84]**. For more recent information refer to **[Smith91]**, which provides a comprehensive bibliography up to 1990.

1.2 What Is FunnelWeb?

FunnelWeb is a particular literate programming system that is implemented by a single C program. FunnelWeb takes as input a single .fw **input file** and writes one or more **product files** and a **documentation file** (see below).



Architecture of FunnelWeb.

In literate programming systems, it is usual to refer to the product file as a "program file". However, as FunnelWeb is a general tool that can be used to prepare all sorts of text files that are not computer programs, the more generic term "product file" was chosen. Product files should be carefully distinguished from the term **output files** which refers to all of the output files produced by FunnelWeb.

FunnelWeb is distinguished by the following characteristics:

> **Simplicity:** A governing design goal of FunnelWeb is to provide a *simple* tool that could be easily learnt and completely mastered. This manual is thick because it is comprehensive and lingers on the ways in which FunnelWeb can be used. The tool itself is quite simple.
>
> **Reliability:** Another design goal is to provide a tool that will protect the user as much as possible from silly errors.

Macro preprocessors are notorious for causing obscure errors. Every attempt has been made in FunnelWeb to keep the syntax robust. For example, in FunnelWeb the syntax of macro calls has been purposely designed to be highly visible so that the reader is always aware when the macro facility is being invoked.

Language and Typesetter Independence: Unlike Knuth's original Web system which was specific to the Pascal programming language**[BSI82]** and the TeX typesetting language**[Knuth84]**, FunnelWeb strives to be language and typesetter independent. FunnelWeb is completely language independent. FunnelWeb input files can be typesetter independent too, and FunnelWeb can generate documentation in TeX and HTML formats.

Portability: FunnelWeb has been written in the C programming language with great emphasis on portability. FunnelWeb currently runs on the Sun, OpenVMS, IBM PC, and Mac.

Controllable: FunnelWeb is an extremely controllable tool. To protect users' investment in source files constructed in the FunnelWeb macro language, the C source code to FunnelWeb has been released under a GNU General Public License (GPL). This means that it will always be available to everyone. Furthermore, license has been granted for the FunnelWeb User's Manual to be copied freely so long as they are not modified. All this means that FunnelWeb is not going to disappear suddenly.

A Production Tool: Above all, FunnelWeb has been designed to be a production tool and every effort has been made to ensure that it will operate effectively in a professional environment. FunnelWeb is "open" and portable. There is a comprehensive user manual. Its error messages are comprehensive. It is fast. Finally, it has been designed with the experience of three years of using FunnelWeb V1.



1.3 The Name FunnelWeb

The name "FunnelWeb" was chosen because it contains the name "WEB", which is the name of Knuth's system. It was also chosen because it has a distinctly Australian flavour.

Funnel-web spiders are found in Northern and Eastern Australia. They are about three to four centimetres long and are very poisonous. The Sydney Funnel-web spider (*Atrax robustus*), common in Sydney, has caused the most trouble and has been responsible for several deaths. Funnel-web spiders love to crawl into temporarily discarded shoes where they later react in a hostile manner to an unsuspecting foot. They are known to hang on once they sink their fangs in. Funnel-web spiders derive their name from the shape of their webs which are horizontally-aligned narrowing tubes, open at one end**[ANZE]**.

The Funnel-web spider, like the tiger snake and the white pointer shark, is secretly regarded by Australians as a kind of national treasure.

F is for Funnel-web
Our furry-legged foe.
He sleeps in your slipper
And breakfasts on toe.
--- One verse from *A Megastar's Mantras: Things that Mean a Lot to Me* ,
by Dame Edna Everage**[Humphries91]**.



Webmaster

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

1.4 Using These Tutorials

Much of the rest of this manual consists of introductory tutorials on FunnelWeb. Ideally you should have a working version of FunnelWeb in front of you so that you can try out the examples yourself. There is no need to try all the examples, so long as you type in enough to feel comfortable with what you are reading.

For best effect, you should create a new, temporary, empty directory in which to experiment with FunnelWeb. That way, it will be more obvious when FunnelWeb creates an output file. You can either type in the examples in this chapter directly, or copy and paste them directly from this manual or the FunnelWeb test suite. The test files called ex01.fw through ex16.fw and hi01.fw through hi10.fw contain the examples in the manual.

If you do not yet have an installed copy of FunnelWeb, refer to the main FunnelWeb web for full details on how to obtain and install a copy of FunnelWeb. If you are not sure if you have an installed copy, try invoking FunnelWeb by giving the command "`fw`". If this yields an error such as "command not found" then you do not have a properly installed version of FunnelWeb.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

www.ross.net

F W

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

1.5 A Hello World Document

Just as one starts the process of learning a new programming language with a "Hello World" program, when learning FunnelWeb, you can start with a "Hello World" document. And here it is! Edit a text file called hello.fw and put the following text in it. (Note: The second character is the letter "Oh", not the digit "Zero").

```
@O@<hello.txt@>@{Hello World@+@}
```

To "run" this "program", invoke FunnelWeb using the "`fw`" command as follows.

```
fw hello
```

If this command doesn't work, then chances are that FunnelWeb has not been installed on your machine. Refer to the main FunnelWeb web for full details on how to obtain and install a copy of FunnelWeb.

There should be no errors. If there are, have a look at the listing file hello.lis, which should contain an explanation of the error, and compare the area in the file where the error occurred with the text above. If there are no errors, you will find that the following two files have been created.

```
hello.lis   - The LISTING file.
hello.txt   - The PRODUCT file.
```

Take a look at hello.txt. It should contain a single line with the text Hello World. Let's take another look at the input file.

```
@O@<hello.txt@>@{Hello World@+@}
```

The whole structure of the input file is controlled by "@", called the **special character**, which introduces **special sequence**s. A scanner's-eye view of the command line looks like this:

```
@O  @<  "hello.txt"  @>
@{  "Hello World"  @+  @}
```

The @ character controls everything. In this file we have six different special sequences that together form a single macro definition. The @< and @> delimit the name of the macro. The @O signals the start of the macro definition and indicates that the macro is to be connected to a product file

with the same name as the macro (This is is why we got a product file when we ran FunnelWeb). The @{ and @} delimit the body of the macro. Finally, the @+ instructs that an end of line sequence should be inserted at that point in the product file.

If you think this syntax looks messy, then you're right. It *is* messy. FunnelWeb *could* have employed a "simpler" notation in which more of the @ sequences were eliminated. For example:

```
Warning: This example is NOT legal FunnelWeb.

#hello.txt{Hello World+}
```

However, if such a syntax were used, the user (you!) would have to remember that # starts a new macro. You would also have to remember that the characters } and + cannot be used in a macro body without a fuss. And so on. FunnelWeb is messier, but provides one simple rule: *Nothing special happens unless the special character @ appears.*

This means that in FunnelWeb, you can look at large blocks of text in the confidence that (unlike for the C pre-processor) there are no macro calls hidden in there. If there were, there would be an @ character! (The only exception to this rule occurs where the user has explicitly changed the special character using the @= special sequence).

Let's take another look at the hello world program.

```
@O@<hello.txt@>@{Hello World@+@}
```

In its current form, it consists of a single macro definition. This definition, while completely valid on its own, only represents half the power of FunnelWeb. In fact you could say that it is a "Hello Northern Hemisphere Program". To turn it into a proper FunnelWeb "Hello World" program, we need to add some documentation.

A FunnelWeb input file consists of a sequence of macro definitions surrounded by a sea of documentation which is just ordinary text. Modify your hello world document so that it looks like this:

```
This hello world document was
created by -insert your name here-.

@O@<hello.txt@>@{Hello World@+@}

It writes out a file called hello.txt
containing the string ``Hello World''.
```

Now run it through FunnelWeb, but this time, add a +t to the command line.

```
fw hello +t
```

If all goes well, you should find that you now have

```
hello.lis   - A LISTING       file.
hello.tex   - A DOCUMENTATION file (in TeX format).
hello.txt   - A PRODUCT       file.
```

Take a look at hello.txt. You will find that it is identical to the hello.txt of the previous run. Only macro definitions affect the product files that FunnelWeb produces (as a result of @O macro definitions). The surrounding documentation has *no* effect. In contrast, the new file, hello.tex (have a look at it now) which was created as a result of your adding the +t option contains a fairly full representation of the input file. Whereas hello.txt is the *product file* of FunnelWeb, hello.tex is the *documentation file* .

Try typesetting the documentation file now using the TeX typesetting program. Then print it. The following commands are an example of the sort of commands you will have to give to do this.

```
tex hello                ! Typeset the doc.
lpr -Pcslw -d hello.dvi ! Print the typeset doc.
```

If you don't have TeX, you can generate an HTML documentation file instead. Here's how:

```
fw hello +u
```

The documentation should consist of single page containing the two lines of documentation along with a typeset representation of the macro. At this point, you have exercised the two main aspects of FunnelWeb. Starting with an input file containing macros (or in this case macro) and documentation, you have successfully generated a product file based on the macros, and a documentation file, based on the entire document.

The next two sections focus on FunnelWeb's macro facilities and its typesetting facilities. By tradition, the generation of program files from a literate text is called **Tangling**, and the generation of typeset documentation is called **Weaving**. In FunnelWeb, these two functions are aspects of a single computer program. However, in Knuth's WEB system, the two functions are embodied in two separate computer programs called Tangle and Weave, presumably because, as everyone knows, "it takes two to Tangle".

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

2 Macro Facilities Tutorial



2.1 Simple Macros

The original "Hello World" program consisted of a single macro definition.

```
@O@<hello.txt@>@{Hello World@+@}
```

In fact, this is a rather exceptional macro, as it causes its expansion to be written to a product file. The @O (for **O**utput) signals this. In FunnelWeb, most macros are defined using @$. This results in a macro that does not generate a product file, but which can be called in other macros (including @O macros). Let us expand the hello world program to include some other macros.

```
@O@<hello.txt@>@{@<Greetings@>@+@}

@$@<H@>==@{Hello@}
@$@<W@>==@{World@}

@$@<Greetings@>==@{@<H@> @<W@>@}
```

Type in the file and run it through FunnelWeb using the command:

```
fw hello
```

The product file (result.out) should look like this:

```
Hello World
```

This short program illustrates some of the features of ordinary macros in FunnelWeb. Consider the @O macro. Instead of containing straight text ("Hello World"), it now contains the macro call @<Greetings@>. A FunnelWeb macro can be called from within the body of another macro just by giving the macro name delimited in @< and @>.

At the bottom of the file is the definition of the @<Greetings@> macro. The definition is similar to the definition of hello.txt except that it starts with @$ to indicate that no product file is desired from this macro (directly). It also employs the optional == syntax which has no semantic impact,

but can be used to make definitions clearer. The body of the @<Greetings@> macro consists of calls to the H and W macros which are defined immediately above.

Note that the macros are not constrained to be defined in any particular order. One of the main features of literate programming tools is that they allow the different parts of the text document being developed (usually a computer program) to be layed out in any order. So long as there is a definition somewhere in the input file for every macro call, FunnelWeb will sort it all out.

In fact, FunnelWeb's macro facility is very simple. Unlike many macro preprocessors which allow macros to define other macros, FunnelWeb completely finishes parsing and analysing the macros in the input file before it starts expanding them into product files. Other preprocessors allow macros to be redefined like variables (as in, say, TeX) taking on many different values as the macro pre-processor travels through the input file. In contrast, FunnelWeb has no concept of "different times" and treats the input as one huge static orderless, timeless, collection of definitions. In FunnelWeb, there is only ever one time, and so there can only ever be one value/definition for each macro.

2.2 Number of Times Called

So far we have seen only tiny, degenerate input files. The next example moves up to the level of "trivial", but starts to convey the flavour of the way FunnelWeb can be used in practice. Normally, there would be documentation text appearing between the macros, but this has been omitted so as to keep the focus on the macros themselves. Although the next example is much longer than the previous example, the only new construct is @- which can appear only at the end of a line, and suppresses it, preventing it from appearing in the text. The @- construct allows the text of a macro to be aligned at the left margin, rather than having the first line hanging at the end of the @{. FunnelWeb could have been set up so that this end of line marker was suppressed. However, it would have been a special case that would have broken the very memorable rule "the text of a macro is the text appearing between the @{ and @}".

Type the following text into the file hello.fw and run it through FunnelWeb. The file contains some intentional errors so be sure to type it in exactly and worry only if FunnelWeb *doesn't* generate some errors.

```
@O@<hello.c@>==@{@-
@<Include Files@>
@<Include Files@>
@<Main Program@>
@}

@$@<Main Program@>==@{@-
main()
{
 doit();
}
@}

@$@<Subroutine@>==@{@-
void doit()
{
 int i;
 for (i=0;i<10;i++)
   {
    @<Print@>
```

```
    @<Print@>
   }
}@}

@$@<Print@>==@{@-
printf("Hello World!");
printf("\n");@}

@$@<Scan@>==@{scanf@}

@$@<Include Files@>==@{@-
#include <stdio.h>
#include <stdlib.h>@}
```

What happened? Well, if you haven't typed the file in properly, you may get some miscellaneous syntax errors. Fix these before continuing. If the file has been correctly typed, you should be faced with some error messages to do with the number of times some of the macros are called.

By default, FunnelWeb insists that each macro defined is invoked exactly once. However, the file above defines macros that are used more than once and a macro that is not used at all. Let us examine the errors.

First, we see that FunnelWeb has alerted us to the fact that the Include Files macro has been called twice. Once alerted to this, a quick look at the program convinces us that calling the macro twice is a mistake, and that one of the calls should be deleted.

Second, we note that FunnelWeb has alerted us to the fact that the @<subroutine@> macro is never called. Again, a quick look at the program tells us that this is a mistake (and a very common one in the use of FunnelWeb), and that a call to the @<subroutine@> macro should be inserted just above the call to the @<Main Program@> macro in the definition of @<hello.c@>.

These two cases demonstrate why these checks have been placed in FunnelWeb. It is nearly always acceptable for a macro to be called once. However, if a macro is not called at all, or called more than once, this is often a sign that the user has made a mistake.

These checks have a dark side too. In addition to the errors mentioned above, FunnelWeb has generated two similar errors that do not help us.

First, we are alerted to the fact that the @<print@> macro has been called twice. Clearly, in this case, this is not a problem, and so here FunnelWeb's fussiness is a nuisance.

Second, we are alerted to the fact that the @<scan@> macro has never been called. Like the @<print@> macro, this macro was defined as a notational convenience, and clearly it does not matter here if it is not used. Again, FunnelWeb is being a nuisance.

The four cases above demonstrate the light and dark side of FunnelWeb's insistence that each macro be called exactly once. To resolve the conflict without reducing the strength of the checking, FunnelWeb provides two special sequences @Z (for **Z**ero) and @M (for **M**any) that can be attached to macro definitions. Presence of the @Z tag allows the designated macro to be called zero times. Presence of the @M tag allows the designated macro to be called more than once. A single macro may carry both tags. It is always true that all macros are allowed to be called exactly once.

Here is the revised program with the errors fixed, by eliminating or adding macro calls, or by adding tags. Try processing the file now. There should be no errors.

```
@O@<hello.c@>==@{@-
@<Include Files@>
@<Function@>
@<Main Program@>
@}

@$@<Main Program@>==@{@-
main()
{
 doit();
}
@}

@$@<Function@>==@{@-
void doit()
{
 int i;
 for (i=0;i<10;i++)
   {
    @<Print@>
    @<Print@>
```

```
    }
}@}

@$@<Print@>@M==@{@-
printf("Hello World!");
printf("\n");@}

@$@<Scan@>@Z==@{scanf@}

@$@<Include Files@>==@{@-
#include <stdio.h>
#include <stdlib.h>@}
```

2.3 Indentation

The body of the print macro of the previous example contains two lines of text. A literal substitution of this macro's body in its context would result in:

```
   {
    printf("Hello World!");
printf("\n");
    printf("Hello World!");
printf("\n");
   }
```

But instead, it comes out as (have a look at this part of hello.c now):

```
   {
    printf("Hello World!");
    printf("\n");
    printf("Hello World!");
    printf("\n");
   }
```

The explanation is that FunnelWeb indents each line of multiline macros by the level of indentation at the point of call. This means that, as in the case above, program texts, which are usually highly indented, come out looking "right".

In other circumstances, where the model of the text is one dimensional, FunnelWeb's indentation could become an impediment or even a danger. In these cases, it can be switched off by including the FunnelWeb **pragma** line

```
@p indentation = none
```

anywhere in the input file.

One of the design goals of FunnelWeb is to allow the user total control over the product files. This contrasts with the approach of Knuth's WEB system **[Knuth83]** (upon which FunnelWeb is based), which mangles the input text at the Pascal program syntax level, truncating identifiers, converting the text to upper case, and paragraphing text. Here is an example of part of a Pascal program produced by WEB (from page 14 of **[Knuth83]**):

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

```
IF R=0 THEN XREF[P]:=XREFPTR ELSE
XMEM[R].XLINKFIELD:=XREFPTR;END;{:51}
{58:}FUNCTION IDLOOKUP(T:EIGHTBITS):NAMEPOINTER;
LABEL 31;
VAR I:0..LONGBUFSIZE;H:0..HASHSIZE;K:0..MAXBYTES;
W:0..1;
L:0..LONGBUFSIZE;P:NAMEPOINTER;BEGIN
L:=IDLOC-IDFIRST;{59:}
H:=BUFFER[IDFIRST];I=IDFIRST+1;
WHILE I<IDLOC DO BEGIN H:=(H+H+BUFFER[I])MOD
HASHSIZE;I=I+1;END{:59};
```

Knuth's theory is that the program generated by a literate programming system should be treated as object code and hence should look like object code too. While this may be an admirable approach in the long run, the present programming environment is one of faulty compilers and buggy tools. The FunnelWeb view is that, in this environment, the programmer needs all the help he can get and that therefore he should be allowed total control over the product file. Another reason for FunnelWeb's providing total control over the product file, is that FunnelWeb is intended to be target language independent, and so even if Knuth's view were adopted, it would not be clear what a legitimate transformation of the text could be.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

2.4 Additive Macros

Sometimes it is convenient to build up the definition of a macro in stages throughout the input file. In FunnelWeb, this can be done using an **additive macro**. An additive macro is identical to an ordinary macro except that

1. It has += instead of ==.
2. It can be defined in one or more parts throughout the input file. The definition of the macro is the concatenation of all the parts in the order in which they appear.

The following example shows how additive macros can be used to scatter and regroup information, in this case assisting in the lucid construction of a data abstraction in a language (Pascal) that does not support them explicitly.

```
@!*******************************

@O@<prog.pas@>==@{@-
program adt(input,output);
@<Types@>
@<Variables@>
@<Procedures@>
begin startproc; end.
@}

@!*******************************

@$@<Types@>+=@{@-
type buffer_type =
   record
   length : integer;
   buf : array[1..100] of char;
   end;
@}

@$@<Variables@>+=@{@-
bigbuf : buffer_type;
@}

@$@<Procedures@>+=@{@-
procedure buf_init (var b : buffer_type                )
   {Body of buf_init}
procedure buf_add  (var b : buffer_type;     ch : char)
   {Body of buf_add}
procedure buf_get  (var b : buffer_type; var ch : char)
```

```
   {Body of buf_get}
@}

@!*****************************

@$@<Types@>+=@{@-
type complex_type = record r,i : real; end;
@}

@$@<Procedures@>+=@{@-
procedure cm_set (var c: complex_type; a,b: real)
   {Body of cm_set}
procedure cm_add (a,b: complex_type; var c: complex_type)
   {Body of cm_add}
{Other procedures and functions}
@}

@!*****************************

{...more pieces of program...}

@!*****************************
```

It is important to remember that the definition of each macro does not change throughout the input file. FunnelWeb parses the entire input file and assembles all the macro definitions before it even starts to expand macros. As a result, each additive macro can only have one definition, and that definition is the concatenation of all its parts.

The example above shows how additive macros can be used to rearrange the presentation of a computer program in the order in which the user wishes to discuss it rather than the order in which the compiler requires that it be consumed. It is easy, however, to abuse the feature of additive macros. In many cases, the same effect can be obtained more clearly by replacing each part of an additive macro in-situ using uniquely named non-additive macros, and then collect them together as a group at the point where the additive macro is called. Doing this is more work, and is more error prone, but can result in a clearer exposition. The following program illustrates this alternative approach.

```
@!*****************************

@O@<prog.pas@>==@{@-
program adt(input,output);
@<Types@>
@<Variables@>
@<Procedures@>
begin startproc; end.
@}
```

```
@$@<Types@>==@{@-
@<Buffer type@>
@<Complex type@>
@}

@$@<Variables@>==@{@-
@<Buffer variable@>
@}

@$@<Procedures@>==@{@-
@<Buffer procedures@>
@<Complex procedures@>
@}

@!*****************************

@$@<Buffer type@>==@{@-
type buffer_type = record
                   length : integer;
                   buf : array[1..100] of char;
                   end;
@}

@$@<Buffer variable@>==@{@-
bigbuf : buffer_type;
@}

@$@<Buffer procedures@>==@{@-
procedure buf_init(var b : buffer_type)
   {Body of buf_init}
procedure buf_add(var b : buffer_type; ch : char)
   {Body of buf_add}
procedure buf_get(var b : buffer_type; var ch : char)
   {Body of buf_get}
@}

@!*****************************

@$@<Complex type@>==@{@-
type complex_type = record r,i : real; end;
@}

@$@<Complex procedures@>+=@{@-
procedure cm_set(var c: complex_type; a,b : real)
   {Body of cm_set}
procedure cm_add(a,b : complex_type; var c: complex_type)
```

```
   {Body of cm_add}
{Other procedures and functions}
@}

@!*****************************

{...more pieces of program...}

@!*****************************
```

One of advantages of FunnelWeb (and literate programming in general) is that (as shown above) it allows the user to lay out the program in whatever order is desired with near total independence from the ordering requirements of the target programming language.

Additive macros are allowed to be tagged with @Z and @M just as other macros can, but the tags must appear only on the first definition of the macro. Additive macros cannot be connected directly to product files.

2.5 Parameterized Macros

No self-respecting macro preprocessor would be complete without some form of macro parameterization, and FunnelWeb is no exception. FunnelWeb allows each macro to have from zero to nine formal parameters named @1, @2, @3, @4, @5, @6, @7, @8, and @9.

To define a macro with one or more parameters, insert a formal parameter list just after the macro name in the macro definition. Because macro parameters have fixed names (@1...@9), there is no need to specify the names of formal parameters in the formal parameter list. All that need be conveyed is how many parameters the macro has. Here is an example of the definition of a macro having three parameters:

```
@$@<While loop@>@(@3@)@M==@{@-
@1
while (@2)
  {
   @3
  }
@}
```

To call a parameterized macro, an actual parameter list must be supplied that contains exactly the same number of actual parameters as there are formal parameters in the definition of the macro being called. An actual parameter list is delimited by @(and @), and parameters are *separated* by @,. The actual parameters themselves are general FunnelWeb expressions (see the FunnelWeb Reference Manual for the exact syntax) and can be inserted into the list directly or can be delimited by @" so as to allow some white space to assist in formatting the actual parameters. Here are some examples of calls of the While loop macro defined above.

```
@! First form of actual parameters
@! without whitespace and double quotes.
@<While loop@>@(@-
x=1;@,x<=10@,printf("X=%u\n",x);@)

@! Second form of actual parameters. The double
@! quotes allow non-active whitespace that helps
@! to lay out the actual parameters neatly. This
@! call is functionally identical to the one above.
@<While loop@>@(
   @"x:=1;@" @,
```

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

```
   @"x<=10@" @,
   @"printf("X=%u\n",x);@" @)

@! The two forms can be mixed in a single call.
@<While loop@>@(x=1;@,x<=10@,
             @"printf("X=%u\n",x);@" @)
```

A few rules about parameterized macros are worth mentioning. Macros that do not have any parameters must have no formal or actual parameter lists. Additive macros can have parameters, but the formal parameter list must appear in the first definition part only.

Here is another example of the use of parameterized macros. This time, parameters and macro calls are used in a FunnelWeb input file that constructs an O(n) representation of a song whose full size is O(n^2) in the number n of unique lines.

```
@O@<Twelve_bugs.txt@>==@{@-
The Twelve Bugs of Christmas
----------------------------
@<Verse@>@(@"first@"    @,@<1@>@)
@<Verse@>@(@"second@"   @,@<2@>@)
@<Verse@>@(@"third@"    @,@<3@>@)
@<Verse@>@(@"fourth@"   @,@<4@>@)
@<Verse@>@(@"fifth@"    @,@<5@>@)
@<Verse@>@(@"sixth@"    @,@<6@>@)
@<Verse@>@(@"seventh@"  @,@<7@>@)
@<Verse@>@(@"eighth@"   @,@<8@>@)
@<Verse@>@(@"ninth@"    @,@<9@>@)
@<Verse@>@(@"tenth@"    @,@<A@>@)
@<Verse@>@(@"eleventh@" @,@<B@>@)
@<Verse@>@(@"twelfth@"  @,@<C@>@)

This song appeared in the internet newsgroup
rec.humor.funny on 24-Dec-1991. It was contributed
by Pat Scannell (scannell@@darkstar.ma30.bull.com).
@}

@$@<Verse@>@(@2@)@M==@{@-
For the @1 bug of Christmas, my manager said to me
     @2
@}

@$@<1@>@M==@{@-
See if they can do it again.@}
@$@<2@>@M==@{@-
```

```
Ask them how they did it and@+@<1@>@}
@$@<3@>@M==@{@-
Try to reproduce it@+@<2@>@}
@$@<4@>@M==@{@-
Run with the debugger@+@<3@>@}
@$@<5@>@M==@{@-
Ask for a dump@+@<4@>@}
@$@<6@>@M==@{@-
Reinstall the software@+@<5@>@}
@$@<7@>@M==@{@-
Say they need an upgrade@+@<6@>@}
@$@<8@>@M==@{@-
Find a way around it@+@<7@>@}
@$@<9@>@M==@{@-
Blame it on the hardware@+@<8@>@}
@$@<A@>@M==@{@-
Change the documentation@+@<9@>@}
@$@<B@>@M==@{@-
Say it's not supported@+@<A@>@}
@$@<C@>@M==@{@-
Tell them it's a feature@+@<B@>@}
```

2.6 Library Macros

FunnelWeb provides a library macro feature that allows you to redefine macros. Normally, FunnelWeb will generate an error if you attempt to define a macro of a particular name more than once. However, if you attach a different number of @L markers (up to five) to each definition, FunnelWeb accepts the multiple definitions, and at tangle time uses the definition with the least number of @Ls. For example:

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

In this example, the ugly duckling macro will expand to swan because the swan definition has the least number of @L markers (i.e. the lowest library level).

No two definitions may have the same name and level, but definitions having the same name, but differing levels, are independent of each other and can have different call number constraints. They can even be defined additively, with their multipart definitions interlaced. For example:

```
@$@<ugly duckling@>@L@L+=@{eg@}
@$@<ugly duckling@>@L+=@{sig@}
@$@<ugly duckling@>@L@L+=@{g@}
@$@<ugly duckling@>@L+=@{net@}
```

Here, two macros having the same name (ugly duckling) are defined at library levels one and two. Each of these macros is defined in two additive parts. The first macro is at level two and has the value egg. The second macro is at level one and has the value signet. If this macro name were referenced by another macro, it would be expanded to signet, as this is the value of the definition with the lowest library level.

Uses Of Library Macros

When using FunnelWeb as a macro preprocessor (e.g. for the generation of HTML webs), it's convenient to be able to define include files that contain large numbers of commonly used

macro definitions. However, sometimes, some macros must be redefined. By tagging such macros in the include file using @L, such redefinition is made possible.

2.7 Macro Expansion

One of the strengths of FunnelWeb is that, when writing product files, it does not attempt to evaluate any text expression (e.g. text block, parameter, macro call) in memory and then write the result out. Instead, it always writes out what it is expanding dynamically and directly. This means that you need not fear defining macros that expand to huge amounts of text and then calling those macros in other macros, or passing those huge macros as parameters to other macros. In all cases, FunnelWeb expands directly to the product file, and there can be no danger in running out of memory during expansion (except for running out of stack space and other marginally used resources in pathological cases).

The only thing to remember in this regard is that FunnelWeb always stores the entire *input* file and all included files, in their entirety in memory, for the duration of the run.

Here is an example, that illustrates how robust FunnelWeb is:

```
@! FunnelWeb copes well with the following
@! macro definitions. (Providing that it has
@! a little over ten megabytes of memory).

@O@<woppa.txt@>==@{@-
@<Quote@>@(@<Humungeous@>@)@+@}

@$@<Quote@>@(@1@)==@{"@1"@}

@$@<Humungeous@>==@{@-
...Ten Megabytes of Text...
@}
```

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

2.8 Include Files

FunnelWeb provides a nested include file facility that can be used for a number of purposes. When FunnelWeb runs into a single line containing the special sequence @i followed by a blank, followed by a file name, it reads in the designated file and replaces the line containing the command (including the end of line marker at the end of the line) with the entire contents of the designated file. For example, if there was a file called camera.txt containing the two lines:

```
'Cos I shoot with a camera instead of a gun.
The animals flock to be petted and fed,
```

and another file called poem.fw containing the following four lines

```
I like to go shooting, it's a whole lot of fun,
@i camera.txt
Cos they know my camera isn't loaded with lead.
- RNW, 04-Jan-1991.
```

Then, if FunnelWeb were to process poem.fw, the result would be as if FunnelWeb had read in:

```
I like to go shooting, it's a whole lot of fun,
'Cos I shoot with a camera instead of a gun.
The animals flock to be petted and fed,
'Cos they know my camera isn't loaded with lead.
- RNW, 04-Jan-1991.
```

FunnelWeb expands include files before it starts scanning and parsing the included text. The result is that include files can contain anything that can be found in a FunnelWeb file. The following example illustrates the level at which the include mechanism operates. If main.fw contains

```
@O@<output.dat@>==@{@-
@i inc.fw
This is the text of the sloth macro.
@}
```

and inc.fw contains

```
@<Sloth@>
```

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

```
@}

@$@<Sloth@>==@{@-
```

Then if FunnelWeb were applied to main.fw, it would see:

```
@O@<output.dat@>==@{@-
@<Sloth@>
@}

@$@<Sloth@>==@{@-
This is the text of the sloth macro.
@}
```

which it would process in the normal manner. The only special sequence processing that takes place at a level lower than include files is the processing of the <special>=<newspecial> sequence which changes the special character.

A few other facts about include files are worth mentioning here. Include files inherit the directory specification supplied using the +I command line option. The special character is saved at the start of each include file and restored to its previous value at the end of each include file. Include files can be nested up to ten levels. Recursive included files will always cause an infinite recursion as there is no bottoming out mechanism available. Include files must contain an integer number of lines (i.e. the last line must be terminated with an end of line marker). Once FunnelWeb has seen "`@i `" at the start of a line, it will grab the rest of the line raw and treat it as a file name. There is no place on the line for things like FunnelWeb comments (see later) or extraneous text.

Include files can be used for many purposes, but are particularly useful for hauling in macro libraries.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

3 Typesetting Facilities Tutorial



Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

3.1 Overview

The previous sections of this manual have focused solely on the macro facilities of FunnelWeb (which are more or less covered completely). As a result, the example documents you have seen so far have been gross distortions of "normal" FunnelWeb documents. Normal FunnelWeb documents often contain as much documentation as code. While there are applications where FunnelWeb can be used solely as a macro preprocessor, many applications will use its typesetting facilities as well.

The macro definitions discussed in the macro tutorial completely define the contents of the product files that FunnelWeb will generate. These macro definitions can be arranged in any order, and nothing external to them can affect the contents of the product files. The macros can be thought of as a group of self-contained islands.

Although FunnelWeb will can process the macros all on their own, the full power of FunnelWeb is realized only when the macros are surrounded by a sea of documentation. This sea can take two forms: directives and free text. Some of the directives control things such as the maximum input line length. However, most of them are typesetting directives that affect the printed documentation. Thus a FunnelWeb document can be viewed as a sequence of **macro definitions**, **directives**, and **free text**.

Unlike the product files which consist of unscrambled macro calls, the documentation file is more or less a direct representation of the input file. Each part of the input file appears in the documentation file in the order in which it appears in the input file. However, each different kind of part is typeset [Note: Here the term "typeset" is used loosely to refer to FunnelWeb's generation of typesetter commands for each construct in the input file. Strictly, the term should be used only to describe the actions of a typesetter program (e.g. TeX).] in a different manner. Macros are typeset in a particular style, with the macro body appearing in tt font (see some FunnelWeb printed documentation for an example). Typesetter directives have specific defined effects (more later). Free text is typeset exactly as it is, except that each block of text between blank lines is filled and justified as a paragraph.

The following example demonstrates how all this works. Type in the following as example.fw and run it through FunnelWeb with the command "`fw example +t`". The "`+t`" instructs FunnelWeb to generate a documentation file called example.tex. Run the file through TeX and print it. Examine the files example.out and example.tex.

```
You are reading some free text before the
macro. Free text can consist of any text
(not containing the FunnelWeb special
character) including typesetter commands
such as $, %, #, and TeX which
will be typeset to appear exactly as
they do in the input file!
Look out! Here comes a macro!

@O@<example.out@>==@{@-
```

```
This text is part of
a macro definition.
@}

This is free text following the macro. This
sentence contains two @{inline@} typesetter
@/directives@/. Now here is a non-inline
typesetting directive.

@t new_page

This sentence will appear on the next page.
```

At the top of the example.tex documentation file will be a set of TeX macro definitions. The TeX code corresponding to the input above appears at the end of the file. It should look something like this.

```
You are reading some free text before the
macro. Free text can consist of any text
(not containing the FunnelWeb special
character) including typesetter commands
such as \$, \%, \#, and \TeX{} which
will be typeset to appear exactly as
they do in the input file!
Look out! Here comes a macro!

\fwbeginmacro
\fwfilename{example.out,1}\fwequals
\fwodef \fwbtx[This text is part of
a macro definition.
]fwetx=%
\fwcdef
\fwbeginmacronotes
\fwisafile{This macro is attached to an output file.}
\fwendmacronotes
\fwendmacro

This is free text following the macro. This sentence contains
two \fwlit{inline} typesetter \fwemp{directives}.
Now here is a non-inline typesetting directive.

\fwnewpage

This sentence will appear on the next page.
```

The following points explain the example.tex file.

> **You don't have to know TeX:** If you don't know TeX, don't pay too much attention to this section. You don't need to know TeX to use FunnelWeb.
>
> **In order:** FunnelWeb has merely transformed the input. It hasn't rearranged it.

Free text: Most of the free text has been simply copied over. The TeX typesetter justifies and fills all paragraphs fed to it by default, so most of the text has just been copied verbatim.

TeX codes: The characters and sequences which TeX treats as special have been neutralized in the documentation file. For example, "$" has become "\$". By default, FunnelWeb allows the user to write any text as free text and not have to worry about accidentally invoking typesetter features.

fw sequences: The fw sequences (e.g. \fwbeginmacro) invoke TeX macros defined earlier in the documentation file (and not shown here).

The macro: The macro is typeset using a set of predefined TeX macros. See the printed documentation to see what this looks like on paper.

Typesetter directives: Unlike the TeX command sequences (which were neutralized), the FunnelWeb typesetter directives turn into TeX macro calls. For example, "`@{inline@}`" became "`\fwlit{inline}`".

FunnelWeb can also generate documentation in HTML form. Just replace the +t option with the +u option.

In summary, FunnelWeb produces typeset documentation that transforms, but does not reorder, the input file. Macros are typeset in a specific style. FunnelWeb typesetter directives have particular well-defined effects. Free text is filled and justified, but will otherwise appear in the printed documentation exactly as it appears in the input file.

3.2 Typesetter Independence

FunnelWeb encourages typesetter independence by neutralizing all TeX (or HTML) control sequences before writing them out. The result is that you don't have worry about upsetting or depending on TeX by accidentally including some special character or sequence. By default your input file is **typesetter independent**.

This scheme differs from other literate programming tools (including very early versions of FunnelWeb) which copy their free text directly to the documentation file, the justification being that the programmer can use the full power of the typesetter language to describe the program. The disadvantages of doing this are first that the programmer is required to know the typesetting language and second that the input file becomes typesetter dependent. FunnelWeb avoids these problems by nobbling the free text be default.

However, FunnelWeb does provide a trapdoor for those who want their free text to be fed directly to TeX. To open the trapdoor, simply include one of the following pragmas somewhere in your input file.

```
@p typesetter = tex
@p typesetter = html
```

See the FunnelWeb Reference Manual for more information.

FunnelWeb leaves the degree to which the user wishes to bind a particular document to a particular typesetter up to the user. In some cases, the extra typesetting power may compensate for the lack of portability. However, as a rule, it is best to avoid typesetter-specific commands, so as to allow your input files to be formatted at a later date for different typesetters. FunnelWeb includes a number of its own typesetter commands so as to support typesetter-independent input files. The following sections describe some of these commands. In particular, the next section describes the most powerful FunnelWeb typesetting directives which allow the user to structure the document hierarchically.

3.3 Hierarchical Structure

The tree structure is one of the most effective structuring tools that exists, deriving its power from the principal of divide and conquor. So effective is it that the internal organization of most technical books are tree structures which are concisely summarized in the table of contents. In contrast, computer programs are usually presented as flat sequences of text to be consumed by an anonymous compiler.

In order to bring program documentation up to the structural sophistication commonplace in technical books, FunnelWeb provides five levels of section headings implemented by the five special sequences @A, @B, @C, @D, and @E. These must always appear at the start of a line. @A is the highest level section (e.g. like LaTeX's \chapter) and @E is the lowest level section (e.g. like LaTeX's \subsubsubsection). Section headings can appear anywhere in the free text of a FunnelWeb input file (i.e. anywhere except inside a macro definition).

Each section heading in a FunnelWeb document has an associated name. The name of a section can be provided explicitly by supplying it delimited by @< and @> immediately after the section sequence (e.g. @A), or implicitly by not providing an explicit name, in which case the section takes the name of the first macro defined between the section header in question and the following section header. An error is generated if a section has not been given an explicit name and does not contain any macro definitions. Here are some example headings:

```
@A@<Feed the Penguins and Save the World@>
@B@<Feed the Penguins@>
@C@<Feed the little penguins@>
@C@<Feed the big penguins@>
@B@<Save the World@>
@C@<Save Europe@>
@C@<Save Africa@>

@C This heading hasn't been given an
explicit name, but will inherit the name
Save the rest of the world
from the macro definition below.
```

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

```
@$@<Save the rest of the world@>@Z==@{...@}
```

The feature of having unnamed sections inherit the name of the first macro defined within their scope is present because a common style of writing in FunnelWeb is to have one section per macro definition. Because, under this style, each section describes a single macro, it usually turns out that the macro name makes a good name for the section too. The inheritance mechanism prevents duplication of the name.

Apart from the requirement that each section have an explicit or implicit name and that its special sequence appear at the start of a line, the only other restriction on section headings is that a section heading at level n cannot appear immediately after a section heading at level n-1 or less. In other words, the hierarchy cannot be broken. For example, an @C cannot appear after an @A heading unless there is an intervening @B heading.

```
@A@<The Top Heading@>
@C@<Level C here is not allowed after an A@>
```

This rule extends to the start of the file; if there are any headings at all, the first one must be an @A heading. The following file, while short, is in error.

```
This FunnelWeb input file is in error
because its first section heading
is at level C rather than level A.
@C@<2@>
```

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

3.4 Understanding the Printed Documentation

Type in the following file, and use FunnelWeb and TeX to generate the corresponding printed documentation.

```
@A@<Table of Contents@>

@t table_of_contents

@A@<Macros for Moral Support@>

The following macro contain comments that provide moral
support.

@$@<Programmer's Cheer@>@M==@{
-- Shift to the left!
-- Shift to the right!
-- Pop up, push down!
-- Byte! Byte! Byte!
-- (From "The New Hacker's Dictionary").
@}

The next macro is similar but is
distributed throughout the program.

@$@<Hacker's Cheer@>+=@{
-- Pointer to the left@+@}

@A@<An Extremely Imperative Stack Abstraction@>

@B@<Define the Stack@>

@$@<Hacker's Cheer@>+=@{@-
-- Pointer to the right@+@}

@$@<Stack Type@>@Z==@{@-
type stack = record ... end;@}

@B@<Push the Stack@>

@$@<Hacker's Cheer@>+=@{@-
-- Hack that code@+@}
```

```
@$@<Push Procedure@>@Z==@{@-
procedure push(var b:stack; v:value);
@<Programmer's Cheer@> {...}@}

@B@<Pop the Stack@>

@$@<Hacker's Cheer@>+=@{@-
-- Tight! Tight! Tight!@+@}

@$@<Pop Procedure@>@Z==@{@-
procedure pop(var b:stack);

@<Programmer's Cheer@> {...}@}

@B@<Rough the Stack Up a Bit@>

@$@<Hacker's Cheer@>+=@{@-
-- (RNW, 04-Jan-1991).@+@}

@$@<Rough Procedure@>@Z==@{@-
procedure rough(var b:stack);
@<Hacker's Cheer@> {...}@}

@O@<dummy.txt@>==@{dummy@+@}
```

An examination of the printed documentation reveals a lot about how FunnelWeb's presentation works.

First, notice how the @t typesetter directive at the top of the file has caused a table of contents to appear. This is one of FunnelWeb's typesetting features and is discussed in a later section. The table of contents shows that the sections have been numbered hierarchically.

Now take a look at the typeset macro definitions. Most important are the numbers in square brackets that follow each macro name. As well as numbering the headings *hierarchically* , FunnelWeb *independently* numbers the macro definitions *sequentially* . The first macro definition (for "Programmer's Cheer") is numbered 1. The second (for "Hacker's Cheer") is numbered 2 and so on. Note that it is not macros that are numbered, but macro definitions. The distinction is necessary because some macros (such as the "Hacker's Cheer" macro) are additive. It is important to realize that there is no relationship between the numbers of the headings and the numbers of the macro definitions.

Now take a look at the notes beneath the body of each macro definition. All macro definitions are followed by a note indicating the definitions in which the macro is called. Additive macros have an additional list, listing the definitions in which they are defined.

Finally, take a look at the macro *call* of "Programmer's Cheer" in section 3.2 of the printed documentation. Macro calls are set in slanted roman (so that they can be distinguished from the tt font code) and are followed by the number of the defining macro definition. In this case, the macro was defined in definition 1. Further down, the call to the "Hacker's Cheer" macro indicates that the macro was defined in definition 2. In fact the macro is additive and definition 2 is just the first of many definitions. To list all definitions in a call to an additive macro would be unnecessarily messy.

3.5 Literals and Emphasis

When writing about program code, it is often desirable to be able to indicate that a particular word or phrase be typeset in the same manner as the code being discussed. For example, one might talk about the variable topval or the procedure stack_pop and wish for them to be typeset as they are in this sentence. This, of course, is simple to do using TeX macros, but use of the (more general) FunnelWeb typesetting directives to do the same work has the added benefit of keeping the document portable to other typesetters.

FunnelWeb provides two in-text type modification constructs: @{...@} and @/...@/ where ... is raw text. The @{...@} construct sets the enclosed text in the same manner as the text of macro definitions is set. The @/...@/ construct emphasises its enclosed text in some typesetter-dependent fashion. Typically the emphasised text is set in italics.

Here is an example of how these constructs might be used:

```
The following procedure @{put_sloth@} writes the
@{sloth@} variable to the output file. Note: @/The output
file must be opened for writing at this point or the program
will crash!@/
```



Webmaster

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

3.6 Adding A Header Page

FunnelWeb provides a few typesetter-independent typesetting constructs which are specifically designed for the construction of header pages. These constructs are usually best placed at the top of your input file, but can be placed anywhere the document if desired to create header pages right through. The two main restrictions on these constructs is that the @t must start at the start of a line (which cannot contain comments), and that the constructs cannot appear inside a macro definition. Here is what the top of an input file might look like:

```
@t vskip 40 mm
@t title titlefont centre "Hairy Wombat"
@t title titlefont centre "Simulation"
@t vskip 10 mm
@t title smalltitlefont centre "A Program in Six Parts"
@t title smalltitlefont centre "Simulating the Life of"
@t title smalltitlefont centre "Some Hairy Wombats"
@t vskip 20 mm
@t title normalfont left "By Zqitzypba Ypongslrzz"
@t new_page
@t table_of_contents
@t new_page
```

The @t at the start of each line indicates that each entire line is a typesetter directive. The vskip directive instructs FunnelWeb to skip some vertical space (measured in millimetres). The title directive instructs FunnelWeb to position a string of text on a single line of its own. Options are provided for font and alignment. The first word after title is the font which can be one of (in decreasing order of size) titlefont, smalltitlefont, and normalfont. The second word after title is the desired alignment of the text. The options here are left, right, and centre. The new_page directive instructs FunnelWeb to skip to a new page. Finally, the table_of_contents directive instructs FunnelWeb to insert a table of contents at that point in the text.



3.7 Comments

A FunnelWeb comment commences with the @! sequence and continues up to, but not including, the end of line marker at the end of the line that the comment sequence is on. Comments can be placed on any line except @i include, @p pragma, and @t typesetter directive lines.

The text following the FunnelWeb comment sequence @! will not appear in the product files or the documentation file. It is only for the eyes of those who bother to look at the original .fw input file. Typically FunnelWeb comments are used to describe the way in which particular FunnelWeb constructs are being used. Example:

```
@! This macro is really revolting.
@! Please forgive me. I had to do it!
@$@<Revolt Me@>==@{@-
@#X@(@#Y@(@#Z@,@"@#Z@"@)=
6@,Teapot@,@"@#Q@(45@)@"@,Tiger@)@}
```



Webmaster

www.ross.net

FW

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

4 A Complete Example

To finish off the chapter, a complete example of a FunnelWeb input file is presented. Although unrealistically short, it gives a better idea of what a typical FunnelWeb .fw file looks like.

```
@!----------------------------------------!
@!  Start of FunnelWeb Example .fw File  !
@!----------------------------------------!

@t vskip 40 mm
@t title titlefont centre "Powers:"
@t title titlefont centre "An Example of"
@t title titlefont centre "A Short"
@t title titlefont centre "FunnelWeb .fw File"
@t vskip 10 mm
@t title smalltitlefont centre "by Ross Williams"
@t title smalltitlefont centre "26 January 1992"
@t vskip 20 mm
@t table_of_contents

@A@<FunnelWeb Example Program@>

This  program writes  out each  of the  first @{p@}
powers of  the first  @{n@} integers. These  constant
parameters are located  here so that they  are easy to
change.

@$@<Constants@>==@{@-
n : constant natural := 10;   -- [1,n] numbers?
p : constant natural :=  5;   -- [1,p]) powers?@}

@B Here is  the outline of the  program. This FunnelWeb
file  generates a single Ada output file  called
@{Power.ada@}. The main program consists  of a loop that
iterates once for each number to be written out.

@O@<Power.ada@>==@{@-
@<Pull in packages@>

procedure example is
   @<Constants@>
begin -- example
   for i in 1..n loop
      @<Write out the first p powers@>
```

```
   end loop;
end example;
@}

@B In this section,  we pull in the packages that this
program  needs to run. In fact, all we need is the IO
package so that we can write out the results. To use the IO
package, we first of all need  to haul it in (@{with
text_io@}) and then we need to make all its identifiers
visible at the top level (@{use text_io@}).

@$@<Pull in packages@>@{with text_io; use text_io;@}

@B Here is  the bit that writes out  the first @{p@}
powers of  @{i@}. The power values  are  calculated
incrementally  in  @{ip@}  to  avoid  the  use  of  the
exponentiation operator.

@$@<Write out the first p powers@>@{@-
declare
   ip : natural := 1;
begin
   for power in 1..p loop
      ip:=ip*i;
      put(natural'image(ip) & " ");
   end loop;
   new_line;
end;@}

@!----------------------------------------!
@!   End of FunnelWeb Example .fw File    !
@!----------------------------------------!
```

Summary

FunnelWeb's functionality can be split into two parts: a macro preprocessor, and support for typesetting. The reader should be aware that the examples in this manual, constructed as they have been to demonstrate particular features of FunnelWeb, do not present a realistic picture of the best use of the tool. Only the example above comes close.

---

5 FunnelWeb Hints

This section contains a grab bag of hints about how FunnelWeb can be used. This chapter probably should not be read until you have commenced using FunnelWeb, or at the very least, tried out some of the examples in earlier chapters. Those who find themselves using FunnelWeb frequently should read this chapter at some stage so as to ensure that they are getting the most out of it.

Most of the examples in this chapter have been placed in the FunnelWeb regression test suite. The files to examine are hi01.fw through hi10.fw.



5.1 Macro Names

When using FunnelWeb, the choice of macro names can be as important to the readability of a program as the choice of program identifiers, and it is important that you know the range of options available.

Names are case sensitive and exact matching: Macro names are case sensitive and are matched exactly. The strings used as a macro name at the point of definition and call must be *identical* for the connection to be made.

Names can contain any printable character: FunnelWeb is less restrictive about its macro names than most programming languages are about their identifiers. A FunnelWeb macro name can contain any sequence of printable characters, including blanks and punctuation. Names can start and end with any character. However, names cannot cross line boundaries. The following are all legal macro names:

```
@<This macro expands to some really bad code@>
@<@>
@<453 #$ % --===~~1">>>@>
@<<@>
@<<>@>
@<a b c d e f g@>
@<       !     @>
@<?? ...@>
@<"Who's been hacking MY program" said Father Bear.@>
@<Update the maximum and return for more data@>
```

Names must be no more than a maximum limit in length: Names can be no longer than a predefined maximum length (80). Currently this length cannot be modified without recompiling FunnelWeb.

Typically, macro names will consist of a short English phrase or sentence that describes the contents of the macro.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

5.2 Quick Names

Sometimes a particular macro must be used extremely often. When this happens, it is desirable to make the macro's name as short as possible. The shortest ordinary FunnelWeb macro name is the empty name "@<@>", which is four characters long. Single-character names are five characters long.

To cater for the cases where really short names are needed, FunnelWeb provides a **quick name** syntax that allows one-character macro names to be specified in two less characters. Quick names take the form of the special character, followed by a hash (#) followed by a single character. Examples:

```
@#A     @#|     @#&     @#m
```

This form of macro name has the same syntactic functionality as an ordinary name and can be substituted wherever an ordinary name can be. In fact quick names live in the same namespace as ordinary macro names. For example the quickname @#A is the *same name* (refers to the same macro) as the ordinary name @<A@>.

Because quick names look syntactically "open" (i.e. they do not have a closing@> as ordinary names do), it is best to avoid them except where a macro must be called very often.

5.3 FunnelWeb the Martinet

There are many ways in which a macro preprocessor can cause unexpected difficulties. FunnelWeb seeks to avoid many of these problems by performing a number of checks. This section describes some of the checks that FunnelWeb performs.

Trailing blanks in the input file: Trailing blanks are usually not dangerous, but FunnelWeb disallows them anyway. All trailing blanks in the *input* (.fw file) are flagged as errors by FunnelWeb. FunnelWeb does not flag trailing blanks in any of its output files.

Input line length: FunnelWeb has a maximum input line length. If FunnelWeb reads an input line longer than this length, it flags the line with an error message. The maximum length can be changed using a pragma (see the FunnelWeb Reference Manual).

Product file line length: FunnelWeb watches the length of output lines and all output lines longer than the limit are flagged with error messages. The maximum length can be changed using a pragma. That FunnelWeb polices output lines is very important. Some programs can behave very strangely if they get an input line that is too long (e.g. Fortran compilers can simply ignore text past a certain column!) and once FunnelWeb starts expanding macros using indentation, it is sometimes not obvious how wide the product file will be.

Control characters: The presence of control characters in a text file can result in some confusing behaviour downstream when the file is presented to various programs. Unfortunately, some text editors allow control characters to be inserted into the text rather too easily, and it is all too easy to be tripped up. FunnelWeb prevents these problems by flagging with diagnostics all non-end-of-line control characters detected in the input (.fw) file (even TABs). The result is that the user is guaranteed that product files generated

from FunnelWeb contain no unintentional control characters. This said, FunnelWeb does allow the insertion of control characters in the output file by explicitly specifying them in the text using a @^ control sequence.

Number of invocations: FunnelWeb checks the number of times that each macro is called and issues an error if the total is not one. The @Z (for zero) and @M (for many) macro attributes can be used to bypass these checks.

Recursion: Because FunnelWeb does not provide any conditional constructs, all recursively defined macros must, by definition, expand infinitely*, and are therefore unacceptable. FunnelWeb performs *static* checks to detect recursion, detecting it before macro expansion commences. The user need not fear that FunnelWeb will lock up or spew forth if a recursive macro is accidentally specified. (* Note: A special case exists where there is recursion but no content. In this case, the expansion is finite (the empty string) even though the operation of expanding is infinite. FunnelWeb does not treat this case specially).

5.4 Fiddling With End of Lines

One of the fiddly aspects of programming with FunnelWeb is coping with end of lines. If you want your product file to be well indented without multiple blank lines or code run-ons, you have to spend a little time working out how the end of line markers get moved around.

The rule to remember is that, disregarding the effects of special sequences within a macro body, *the body of a macro consists of exactly the text between the opening @{ and the closing @}* . This text includes end of line markers.

If for example you call a macro in a sequence of code...

```
while the_walrus_is_sleepy do
   begin
   writeln('zzzzzzz');
   @<Wake up the walrus@>
   writeln('Umpharumpha...');
   end;
```

where <wake up the walrus> is defined as follows

```
@$@<Wake up the walrus@>==@{
wake_up_the_walrus(the_walrus);
@}
```

then when <Wake up the walrus> is expanded you will get

```
while the_walrus_is_sleepy do
   begin
   writeln("zzzzzzz");

   wake_up_the_walrus(the_walrus);

   writeln("Umpharumpha...");
   end;
```

The blank lines were introduced by the end on line markers included in the definition of <Wake up the walrus>. A good solution to this problem is to suppress the end of line markers by defining the macro as follows

```
@$@<Wake up the walrus@>==@{@-
wake_up_the_walrus(the_walrus);@}
```

This is the usual form of macro definitions in FunnelWeb files.

In additive macros, this format does not work properly because the end of line that is suppressed by the trailing @} does not get replaced by the end of line at the end of the macro invocation. For example the definition

```
@$@<Wake up the walrus@>+=@{@-
wake_up_the_walrus_once(the_walrus);@}
```

later followed by

```
@$@<Wake up the walrus@>+=@{@-
wake_up_the_walrus_again(the_walrus);@}
```

is equivalent to the single definition

```
@$@<Wake up the walrus@>==@{@-
wake_up_the_walrus_once(the_walrus);@-
wake_up_the_walrus_again(the_walrus);@}
```

Putting the trailing @} on a new line at the end of the macro (except for the last definition part) solves the problem.

```
@$@<Wake up the walrus@>+=@{@-
wake_up_the_walrus_once(the_walrus);
@}
```

later followed by

```
@$@<Wake up the walrus@>+=@{@-
wake_up_the_walrus_again(the_walrus);@}
```

is equivalent to the single definition

```
@$@<Wake up the walrus@>==@{@-
wake_up_the_walrus_once(the_walrus);
wake_up_the_walrus_again(the_walrus);@}
```

Managing end of line markers is tricky, but once you establish a convention for coping with them, the problem disappears into the background.

5.5 Fudging Conditionals

As a macro preprocessor, one facility that FunnelWeb lacks is a conditional facility (such as C's #ifdef). It might, therefore, come as a surprise to know that the FunnelWeb V1 actually had a built in conditional facility. The facility allowed the programmer to specify a construct that would select from one of a number of macro expressions depending on the value of a controlling macro expression.

In three years the construct was never used.

The reason was that conditional constructs could be fudged nearly as easily as they could be used. Because of this, the inbuilt conditional feature was removed in the FunnelWeb V2. Not only did this simplify the program, but is also allowed recursive macros to be detected through static analysis rather than during macro expansion.

There are two basic ways to fudge a conditional. First, the comment facility of the target programming language may be employed. For example, in Ada, comments commence with "--" and terminate at the end of the line. Using this fact, it is easy to construct macros that can be called at the start of each target line and which turn on and off the lines so marked by defining the macro to be the empty string (ON) or the comment symbol (--) (OFF). For example:

```
@A@<Debug Macro@>

The following macro determines whether debug code
will be included in the program. All lines of
debug code commence with a call to this macro and
so we can turn all that code on or off here by
defining this macro to be either empty or the
single-line comment symbol (--). Note
the use of a quick macro name.

@$@#D@M==@{@}  @! Turns the debug code ON.
@! Use this definition to turn
@! the debug code OFF: @$@#D==@{--@}

... then later in the file...

@$@<Sloth incrementing loop@>==@{@-
while sloth<walrus loop
```

```
   @#D assert(sloth<walrus,"Sloth bomb.");
   @#D assert(timer<timermax,"Timer bomb.");
   inc(sloth);
end loop@}
```

The other way to fudge a conditional is to define a macro with a single parameter. A call to the macro is then wrapped around all the conditional code in the program. The macro can then be defined to present or ignore the code of its argument. For example:

```
@A@<Debug Macro@>

The following macro determines whether debug code
will be included in the program. All debug code is
wrapped by a call to this macro and so we can turn
all the debug code on or off here by defining this
macro to be either empty or its parameter.

@$@#D@(@1@)@M==@{@1@}  @! Debug code ON.
@! Use this definition to turn the
@! debug code OFF: @$@#D@(@1@)==@{@}

... then later in the file...

@$@<Sloth incrementing loop@>==@{@-
while sloth<walrus loop
   @#D@(assert(sloth<walrus,"Sloth bomb.");
        assert(timer<timermax,"Timer bomb");@)
   inc(sloth);
end loop@}
```

In languages that allow multi-line comments (e.g. C with /* and */), comments can be used to eliminate the conditioned code rather than absence. For example:

```
@! Comments out the debug code
@$@#D@(@1@)@M==@{/* @1 */@}
```

(Note: If this example were ever actually used, the programmer would have to be careful not to place comments in the argument code. Nested comments in C are non-portable.)

The parameterized macro idea can be generalized to support the choice of more than one mutually exclusive alternative. For example:

```
@A This module contains non-portable code that
```

```
must execute on Hewlett Packard, Sun, and DEC
workstations. The following FunnelWeb macro is
defined to choose between these three. The first
parameter is the HP code, the second is the Sun
code, and the third is the DEC code. Whichever
parameter constitutes the body of this macro
determines which machine the code is being
targeted for.

@$@<Machine specific code@>@(@3@)@M==@{@-
@1@}  @! Configure for HP.

...then later in the file...

@<Machine specific code@>@(
@"get_command_line(comline)@"           @, @! HP.
@"scan_command_line(128,comline);@"     @, @! Sun.
@"dcl_get_command_line(comline,256);@"  @) @! DEC.
```

Of course, this could also be performed using three separate macros. The main advantage of using a single macro is that the mutual exclusivity is enforced. Also, because FunnelWeb ensures that the number of formal and actual parameters are the same, this method lessens the chance that a machine will be forgotten in some places.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

5.6 Changing the Strength of Headings

FunnelWeb provides five heading levels: @A, @B, @C, @D, and @E to which it binds five different typographical strengths. These bindings are static; a level @A heading will always be typeset in a particular font size regardless of the size of the document. The font sizes have been preset to be "reasonable" for a range of document sizes, but may be inappropriate for very small or large documents.

FunnelWeb does not currently provide an "official" way (e.g. a pragma) to change the typesetting strength of headings. This feature might be added in later versions. Meanwhile, a hack is available that will do the job, providing that you do not mind the hack being TeX-specific and probably FunnelWeb-version specific.

Inside the set of TeX macro definitions that FunnelWeb writes at the top of every documentation file are five "library" definitions fwliba...fwlibe which provide five different typesetting strengths for headings. Near the end of the set of definitions, FunnelWeb binds these macros to five other macros fwseca...fwsece which are invoked directly in the generated TeX code to typeset the headings.

```
\def\fwseca#1#2{\fwliba{#1@,#2}}
\def\fwsecb#1#2{\fwlibb{#1@,#2}}
\def\fwsecc#1#2{\fwlibc{#1@,#2}}
\def\fwsecd#1#2{\fwlibd{#1@,#2}}
\def\fwsece#1#2{\fwlibe{#1@,#2}}
```

This means that the typesetting strength of the headings in a FunnelWeb document can be changed by redefining these macros at the top of a FunnelWeb document. For example:

```
@p typesetter = tex
\def\fwseca#1#2{\fwlibc{#1@,#2}}
```

would set @A headings at the same strength as the default strength of @C headings. The typesetter directive is necessary to ensure that the TeX control sequences get through to the documentation file unfiltered.

The following will tone down all headings by two levels (with the @D and @E levels being allocated the default @E typesetting strength because there is nothing weaker).

```
@p typesetter = tex
\def\fwseca#1#2{\fwlibc{#1@,#2}}
\def\fwsecb#1#2{\fwlibd{#1@,#2}}
\def\fwsecc#1#2{\fwlibe{#1@,#2}}
\def\fwsecd#1#2{\fwlibe{#1@,#2}}
\def\fwsece#1#2{\fwlibe{#1@,#2}}
```

These definitions affect only the headings that follow them, and so they should be placed at the top of the FunnelWeb input file.

To change the style of headings in HTML output, just modify the style settings near the top of the HTML file.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

5.7 Efficiency Notes

The following notes are worth keeping in mind when using FunnelWeb.

Memory: When FunnelWeb processes an input file, it reads the entire input file, and all the included files into memory. (Note: If a file is included n times, FunnelWeb keeps n copies in memory). This organization does not pose a constraint on machines with large memories, but could present a problem on the smaller machines such as the PC.

Speed: FunnelWeb is not a slow program. However, it is not particularly fast either. If the speed at which FunnelWeb runs is important to you, then the thing to keep in mind is that FunnelWeb has been optimized to deal efficiently with large slabs of text. FunnelWeb treats input files as a sequence of text slabs and special sequences (e.g. @+) and whenever it hits a special sequence, it has to stop and think. Thus, while a ten megabyte text slab would be manipulated as a single token, in a few milliseconds, a similar ten megabyte chunk filled with special sequences would take a lot longer. If FunnelWeb is running slowly, look to see if the input contains a high density of special sequences. This can sometimes happen if FunnelWeb is being used as a backend macro processor and its input is being generated automatically by some other program.

Macro expansion: When tangling (expanding macros), FunnelWeb never expands a macro expression into memory; it always writes it to the product file as it goes. This is a powerful fact, because it means that you can write macros containing an unlimited amount of text, and pass such macros as parameters to other macros without becoming concerned about overflowing some kind of buffer memory. In short, FunnelWeb does not impose any limits on the size of macro bodies or their expansions.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

5.8 Interactive Mode

As well as having a command line interface with lots of options, FunnelWeb also provides a command language and a mode ("interactive mode") in which commands in the language can be typed interactively. The FunnelWeb command interpreter was created primarily to support regression testing, but can also be useful to FunnelWeb users.

FunnelWeb's command interpreter reads one command per line and can read a stream of commands either from a text file, or from the console. The interpreter can understand over twenty commands. See the FunnelWeb Reference Manual for a full list. However, most of them were designed to support regression testing and will not be of use to the casual user.

The commands that are of greatest use to the casual user are:

```
!                - Ignores the whole line.
EXECUTE fn       - Execute the specified file.
FW options       - Invoke FunnelWeb-proper once.
SET options      - Sets options.
SHOW             - Displays current options.
TRACE ON         - Turns command tracing ON.
QUIT             - Quits FunnelWeb.
```

To distinguish here between invocations of the FunnelWeb program and FunnelWeb runs inside the shell, we call the latter **FunnelWeb proper**. The "FW" command invokes FunnelWeb proper with the specified options which take the same syntax as they do on the command line. The only restriction is that none of the action options can be turned on except "+F" which must be turned on.

The "SET" command has the same syntax as the "FW" command except that it does not allow *any* action options to be specified. It's sole effect is to set default option values for the rest of the run.

The "SHOW" command displays the current default options.

By default, FunnelWeb does not echo the commands that it processes in a script. The "TRACE ON" command turns on such tracing.

These commands can be combined to streamline the use of FunnelWeb. For example, you might wish to create a script called typeset.fws to process a whole group of files.

```
trace on
!This script typesets the whole program.
! Set no listing file, no product files,
! but specify a documentation file
! and specify the directory into which
! it should be placed.
set -L -O +T/usr/ross/typeset/
fw prog1
fw prog2
fw prog3
fw prog4
```

There are a few ways in which this script can be run. The simplest is simply to specify it in the "+X" option of a FunnelWeb invocation. FunnelWeb shellscripts default to "<current_directory>" and ".fws".

```
fw +xtypeset
```

The second alternative is to enter interactive mode.

```
fw +k
```

From there, you can execute the script using:

```
execute typeset
```

Interactive mode could be very useful to those with multiple-window workstations. The user could create a window containing an interactive session of FunnelWeb, and then switch between windows, editing, and executing FunnelWeb proper and other programs.

If you find yourself using the command interpreter a lot, be sure to read about the other commands that are available in the FunnelWeb Reference Manual.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

5.9 Setting Up Default Options

If you do not like FunnelWeb's default settings for its command line options, there are a number of ways in which you can change them.

Define an "alias": Use your operating system "`alias`" facility to create an alias for FunnelWeb containing the desired options. FunnelWeb processes options from left to right, so you can override these defaults later if you wish.

Create a script called "`fwinit.fws`": When FunnelWeb starts up, it executes a script called "`fwinit.fws`" if such a script exists in the current directory. You can use this fact to set options before the run of FunnelWeb proper by creating such a script and placing a single "`set`" command in it containing the desired options. The main trouble with this approach is that the options in the set command will be processed *after* the command line options, which means that you won't be able to override them on the command line.

For example, you might be involved more with presenting programs than with running them, and want FunnelWeb to generate a documentation file by default, but not to produce listing or product files by default. In Unix you could do this with:

```
alias fw fw -L -O +T
```



FunnelWeb Tutorial Manual

5.10 FunnelWeb and Make

The Unix Make program allows a set of dependencies between a set of files to be described, and then uses these dependencies to control the way in which the files are created and updated. Typically, Make is used to control the process of transforming a collection of source code files to one or more executable files. As the use of FunnelWeb implies an extra stage to this process, it is natural to include the transformation of .fw files to source code files as part of the Make process. This is easy to do, but the user should be aware of one aspect of FunnelWeb which can cause problems.

It is often useful, when using FunnelWeb, to create a FunnelWeb .fw file that generates more than one product file. That is, a single .fw file may have many macro definitions connected to product files so that when the FunnelWeb .fw file is processed by FunnelWeb, several files are created. For example, this facility is convenient for placing a single package's .h and .c files within the same FunnelWeb .fw file.

The use of multiple product files, however, provokes a problem with dependencies. Suppose for example that a FunnelWeb prog.fw produces two product files proc.spec (a package specification) and prog.body (a package body). If the package is accessed in the way that packages normally are, it will be quite common for the programmer to want to modify the package body without modifying the program specification. So the programmer will edit the prog.fw file to change the package body. The result of running this through FunnelWeb will be the desired new package body file. However, FunnelWeb will also produce a new package specification product file *even though it may be identical to the previous version!* The result is that the newly created (with a recent file date) specification package file could provoke a huge remake of much of the program in which it resides.

To solve the problem, FunnelWeb includes a command line option (D for Delete), which when turned on (using "+D") causes FunnelWeb to suppress product and documentation files that are identical to the previously existing versions of the same files. For example, if, during a FunnelWeb run, a macro was connected to a product file called x.dat, and the macro expanded to *exactly* the same text as is contained in x.dat then

FunnelWeb would simply *never write the product file* , the file x.dat would be untouched and, as a result, no further Make propagations would take place.

FunnelWeb implements this feature by writing each product file to a temporary file with a temporary file name. It then compares the temporary file with the target file. If the two are identical, it deletes the temporary file. If the two are different it deletes the target file and renames the temporary file to the target file.

Use of the D facility means that the programmer need not be punished (by extra Make propagations) for describing more than one product file in the same FunnelWeb file.

5.11 The Dangers Of FunnelWeb

Like many tools that are general and flexible, FunnelWeb can be used in a variety of ways, both good and bad. One of the original appeals of the literate approach to programming for Knuth, the inventor of literate programming, was that it allows the programmer to describe the target program bottom up, top down, size to side, or chaotically if desired.

The flexibility that the literate style of programming leaves much room for bad documentation as well as good documentation. Years of experience with FunnelWeb has revealed the following stylistic pitfalls which the experienced FunnelWeb user should take care to avoid. (Note: The fact that these faults are listed here does not mean that the author has eliminated them in his own work. Rather, it is mainly the author's own mistakes that have resulted in this list being compiled. The author immediately confesses to several of the faults listed here, most notably that of Pavlov documentation).

Spaghetti organization: By far the worst problem that arises in connection with the literate style occurs where the programmer has used the literate tool to completely scramble the program so that the program is described and layed out in an unordered, undisciplined "stream of consciousness". In such cases the programmer may be using the literate style as a crutch to avoid having to think about structuring the presentation.

Boring organization: At the other extreme, a program may be organized in such a strict way that it is essentially laid out in the order most "desired" by the target programming language. For example, each macro might contain a single procedure, with all the macros being called by a macro connected to a file at the top. In many cases a boring structure may be entirely appropriate, but the programmer should be warned that it is easy to slip into such a normative style, largely forgetting the descriptive structural power that FunnelWeb provides.

Poor random access: Using FunnelWeb, it is quite possible to write programs like novels --- to be read from cover to cover. Sometimes the story is very exciting, with data structures making dashing triumphs and optimized code bringing the story to a satisfying conclusion. These programs can be works of art. Unfortunately, without careful construction, such "novel-programs" can become very hard to access randomly by (say) a maintenance programmer who wishes only to dive in and fix a specific problem. If the entire program is scrambled for sequential exposition, it can be hard to find the parts relating to a single function. Somehow a

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

balance must be struck in the document between the needs of the sequential and of the random-access reader. This balance will depend on the intended use of the program.

Too-interdependent documentation: Sometimes, when editing a program written using FunnelWeb, one knows how to modify the program, but one is unsure of how to update the surrounding documentation! The documentation may be woven into such a network of facts that it seems that changing a small piece of code could invalidate many pieces of documentation scattered throughout the document. The documentation becomes a big tar pit in which movement is impossible. For example, if you have talked about a particular data structure invariant throughout a document, changing that invariant in a small way could mean having to update all the documentation without touching much code. In such cases, the documentation is too interdependent. This could be symptomatic of an excessibly interconnected program, or of an excessively verbose or redundant documenting style. In any case, a balance must be struck between the conversational style that encourages redundancy (by mentioning things many times) and the normalized database approach where each fact is given at only one point, and the reader is left to figure out the implications throughout the document.

Pavlov documentation: By placing so much emphasis on the documentation, FunnelWeb naturally provides slots where documentation "should" go. For example, a FunnelWeb user may feel that there may be a rather unpleasant gap between a @C marker and the following macro. In many cases *no* commentary is needed, and the zone is better left blank rather than being filled with the kind of uninformative waffle one often finds filling the slots of structured documentation written according to a military standards (e.g. MIL-STD-2167A). (Note: This is not a criticism of 2167A, only of the way it is sometimes used). The lesson is to add documentation only when it adds something. The lesson in Strunk and White**[Strunk79]** (p. 23) holds for program documentation as it does for other writing: "Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts. This requires not that the writer make all his sentences short, or that he avoid all detail and treat his subjects only in outline, but that every word tell.".

Duplicate documentation: Where the programmer is generating product files that must exist on their own within the entire programming environment (e.g. the case of a programmer in a team who is using FunnelWeb for his own benefit but must generate

(say) commented Ada specification package files) there is a tendency for the comments in the target code to duplicate the commentary in the FunnelWeb text. This may or may not be a problem, depending on the situation. However, if this is happening, it is certainly worth the programmer spending some time deciding if one or other of the FunnelWeb or inline-comment documentation should be discarded. In many cases, a mixture can be used, with the FunnelWeb documentation referring the reader to the inline comments where they are present. For example:

```
@A Here is the header comment for the list package
specification. The reader should read these comments
carefully as they define a list. There is no need to
duplicate the comments in this text.

@$@<Specification package comments@>==@{@-
-- LIST PACKAGE
-- ============
-- * A LIST consists of zero or more ITEMS.
-- * The items are numbered 1 to N where N
--   is the number of items in the list.
-- * If the list is non-empty, item 1
--   is called the HEAD of the list.
-- * If the list is non-empty, item N
--   is called the TAIL of the list.
-- ...
@}
```

Overdocumenting: Another evil that can arise when using FunnelWeb is to over-document the target program. In some of Knuth's earlier (e.g. 1984) examples of literate programming, each variable is given its own description and each piece of code has a detailed explanation. This level of analysis, while justified for tricky tracts of code, is probably not warranted for most of the code that constitutes most programs. Such over-commenting can even have the detrimental affect of obscuring the code, making it hard to understand because it is so scattered (see "spaghetti organization" earlier). It is up to the user to decide when a stretch of just a few lines of code should be pulled to bits and analysed and when it is clearer to leave it alone.

In the case where there are a few rather tricky lines of code, a detailed explanation may be appropriate. The following example contains a solution to a problem outlined in section 16.3 of the book "The Science of Programming" by David Gries**[Gries81]**.

```
@C@<Calculation of the longest plateau in b@>
```

```
This section contains a solution to a problem
outlined in section 16.3 of the book @/The
Science of Programming@/ by David Gries[Gries81].

@D Given a sorted array @{b[1..N]@} of
integers, we wish to determine the @/length@/ of
the longest run of identically valued elements in
the array. This problem is defined by the
following precondition and postcondition.

@$@<Precondition@>==@{/* Pre: sorted(b). */@}
@$@<Postcondition@>==@{@-
/* Post: sorted(b) and p is the length */
/* of the longest run in b[1..N].      */@}

@D We approach a solution to the problem by
deciding to try the approach of scanning through
the array one element at a time maintaining a
useful invariant through each iteration. A loop
variable array index @{i@} is created for this
purpose. The bound function is @{N-i@}. Here is
the invariant.

@$@<Invariant@>==@{@-
/* Invariant: sorted(b) and 1<=i<=N and   */
/*            p is len of longest run in b[1..i]. */@}

@D Establishing the invariant above in the
initial, degenerate case is easy.

@$@<Establish plateau loop invariant@>@{i=1; p=1;@}

@D At this stage, we have the following loop
structure. Note that when both the invariant and
@{i == N@} are true, the postcondition holds and
the loop can terminate.

@$@<p=len(longest run in sorted b[1..N])@>@{@-
@<Precondition@>
@<Establish plateau loop invariant@>
while (i != N)
  {
   @<Invariant@>
   @<Loop body@>
  }
```

```
@<Postcondition@>
@}

@D Now there remains only the loop body whose
sole task is to increase @{i@} (and so decrease
the value of the bound function) while maintaining
the invariant. If @{p@} is the length of the
longest run seen so far (i.e. in b[1..i]), then,
because the array is sorted, the extension of our
array range to @{b[1..i+1]@} can only result in
an increase in @{p@} if the new element
terminates a run of length @{p+1@}. The increase
can be at most 1. Because the array is sorted, we
need only compare the endpoints of this possible
run to see if it exists. This is performed as
shown below.

@$@<Loop body@>==@{i++; if (b[i] != b[i-p]) p++;@}
```

Where the code is more obvious, it is often better to let the code speak for itself.

```
@C The following function compares two C
strings and returns TRUE iff they are identical.

@$@<Function comp@>==@{@-
bool comp(p,q)
char *p,*q;
{
 while (TRUE)
   {
    if (*p != *q  ) return FALSE;
    if (*p == '\0') return TRUE;
    p++; q++;
   }
}
@}
```

5.12 Wholistic Debugging

Surprising though it may be, FunnelWeb has a role to play in the **debugging** of programs. My experience in programming has led me to the concept of **wholistic debugging**. When many programmers detect a bug, their first reaction seems to be to jump into the debugger where they often spend many hours stepping through endless stretches of code and generally wasting a lot of time.

In contrast, my first reaction when I detect a bug is to realize that **the code must not be in good enough shape if such a bug can arise!** The presence of the bug is taken as symptomatic of the lack of general health of the code. If that bug occurred, why not another? In response to this realization, my reaction is not to enter the debugger, but rather to return to the original code and tend it like a garden, adding more comments, reworking the weaker parts, adding assertions, and looking for faults. In many cases, the search for faults does not even centre on the specific bug that arose, but does tend to focus on the area of code where the bug is likely to be.

The result is often that the original bug is located more quickly than it would have been had the debugger been involved. But even if it isn't, there are other benefits. A programmer who enters the debugger may stay there for hours and still not find the bug. The result is frustration and no positive gain at all. In contrast, by tending to the code, the programmer is making forward progress at all times (the code is constantly improving) even if the bug is not immediately found. At the end of ten hours, the programmer can at least feel that the code is "ten hours better", whereas the debugger freak will likely feel defeated. All this makes code tending better psychologically as well as a more efficient approach to debugging.

I call this technique wholistic debugging, for it is like the difference between conventional and wholistic medicine. Go to a conventional doctor with a headache and he might send off for head X-rays, perform allergy tests and perform many other debugging activities. Go to a wholistic doctor with the same problem and he might look to see if you are fit, assess your mental health, and ask you if your marriage is working. Both approaches are appropriate at different times, but I believe that, on balance, in programming the wholistic approach is not used

enough.

5.13 TABs

FunnelWeb is designed to protect its user from spurious non-printable characters that might creep into a source file, and so it classifies any occurrence of such a character as an error.

FunnelWeb's idea of printable is any character in the range ASCII [32..126]. A number of people have complained about this, saying that they want to include 8-bit characters in their FunnelWeb input files. Currently there is no fix for this except to insert characters explicitly into the output using a @^ sequence (see the FunnelWeb Reference Manual for more details).

In particular, FunnelWeb flags TAB characters in its input file as errors because TABs are essentially invisible control characters with very poorly-defined semantics; it's FunnelWeb's job to keep an eye on the input file to ensure that the programmer doesn't get any nasty surprises and as TABs can cause several kinds of surprises, they are excluded.

A lot of people have complained about this aspect of FunnelWeb. Unfortunately, this issue has not yet been resolved. If you actually *need* TABs in the output file (e.g. for a makefile), then you should insert TABs explicitly using a @^D(009) sequence (see the FunnelWeb Reference Manual for more details).

The following C program removes TABs. The program was kindly donated by John Skaller. Warning: This program doesn't have any error checking.

```
/* UNTAB Program                                                */
/* =============                                                */
/* Author       : John Skaller (maxtal@extro.ucc.su.oz.au) */
/* Organization : MAXTAL P/L 6 Mackay St ASHFIELD 2131.     */
/* Usage        : untab 2 outfile                    */
/* Status       : Public domain.                                */

#include

int main(int argv, char **argc)
{
        int n=2;
        if(argv==2)sscanf(argc[1],"%d",&n);
        int ch=getchar();
        int column=0;
        while(ch!=-1){
                if(ch=='\n'){putchar('\n'); column=0;}
                else if(ch==9){
                        putchar(' '); column++;
                        while(column % n){putchar(' '); column++;}
                }
                else { putchar(ch); column++;}
                ch=getchar();
        }
}
```

5.14 HTML Style

If you specify the +U command line option, FunnelWeb will generate a .html documentation file containing HTML. In the <HEAD> section of this file, FunnelWeb defines the style of the file by including a style directive similar to the following:

```
<STYLE TYPE="text/css">
<!--
A {text-decoration: none}
H1 { font-family: sans-serif; font-size: large }
H2 { font-family: sans-serif; font-size: medium;
     font-weight: bold }
H3 { font-family: sans-serif; font-size: medium }
H4 { font-family: sans-serif; font-size: small }
H5 { font-family: sans-serif; font-size: small }
// -->
</STYLE>
```

If you do not like this style, you can change it simply by editing the .html file and changing the style directive in the <HEAD> section. However, this change will be overwritten the next time you regenerate the HTML file.

Another way to define your own style is to turn off FunnelWeb's automatic "escaping" of special characters with:

```
@p typesetter = html
```

You can then include a style directive in one one of the very early comment parts of your FunnelWeb source file, where it will override the one written by FunnelWeb in the <HEAD> section. The disadvantage is that your FunnelWeb source file becomes typesetter-dependent.



Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

5.15 A FunnelWeb Mode For Emacs

Tony Coates has created a FunnelWeb Emacs mode, which he has made generally available. The mode is written as a FunnelWeb file. To use the mode, click on the link below. A new window should appear containing a plain text file which is the FunnelWeb source for the mode. Save the plain text file on your disk as fwmode.fw and then invoke FunnelWeb using fw fwmode to generate the emacs mode files fw-mode.el and switch-mode.el.

FunnelWeb Emacs Mode FunnelWeb Source File



---

6 Examples of FunnelWeb Applications

Despite (or perhaps because of) its flexibility and simplicity, FunnelWeb can be applied to quite a number of different text processing and documenting problems. This section gives some examples of some of the more interesting real problems that FunnelWeb has solved.



www.ross.net
F W
Reference
Developer
Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking
SEARCH

6.1 Analyzing the Monster Postscript Header File

During my Ph.D. candidature, I determined at one point that it would be very desirable to automatically insert diagrams from the *MacDraw* program on my Macintosh into TeX insertions in my thesis. This would allow diagrams to float around with the text and be printed automatically rather than having to be printed separately and stuck in with real glue. On the face of it, the problem seemed inherently solvable, as the Macintosh could generate PostScript for each diagram and this PostScript could presumably be inserted into the PostScript generated using TeX.

The only trouble was that the Macintosh PostScript code for the diagrams relied on an Apple PostScript header file. This meant that the header file had to be included at the start of the TeX PostScript if the inserted PostScript for the diagrams was to work. Unfortunately, merely including the header file at the top didn't work, and it turned out that a rather detailed analysis of some parts of the Apple header file was required in order to perform the necessary surgery on the header file to make it work. This analysis was severely aggravated by the fact that the PostScript header file was virtually unreadable. Basically it was about 50K of interwoven definitions, that looked as if it had been run through a word processor. There was no way that the code could be understood clearly without some kind of reformatting. Two other aspects of the problem further complicated the analysis:

- The definitions of interest (i.e. the ones causing the problems) were scattered throughout the heaeder file.
- Many definitions could not be moved within the header file. For one or more reasons (e.g. to keep a definition within the activation of a particular dictionary begin and end) it would have been unwise to move the definitions of interest to the same point in the file.

In fact the file was so messy and complicated that, as a rule, it had to be handled with kid gloves. It would have been unwise to re-arrange the definitions or to insert comments.

To my surprise, FunnelWeb provided an unexpected solution to the problem. First I replaced all occurrences of the @ in the header file with @@. Second, I placed the entire header file in a FunnelWeb macro definition connected to a product file. I then processed the file and checked to make sure that the product file was identical to the original file. By doing all this I had placed the header file under FunnelWeb control. I then left the macro definition largely untouched, but replaced the PostScript definitions of interest with FunnelWeb macro calls, moving the actual PostScript definitions into FunnelWeb macro definitions at the end of the FunnelWeb file.

```
@O@<LaserHeader.ps@>==@{@-
Unreadable Postscript code
@<Print routine@>
Unreadable Postscript code
@<Zap routine@>
Unreadable Postscript code
@}

@A This routine looks as if it does this, but really is
does that, blah, blah blah.

@$@<Print routine@>==@{@-
/print { push pop pop push turn around
         and jump up and down and print it} def
@}

@A This routine zaps the...

@$@<Zap routine@>==@{@-
/zap { push pop pop push turn around and
       jump up and down and print it} def
@}
```

Use of FunnelWeb meant that I was able to pluck out the definitions of interest (a very small part of the whole file) and collect them as a group at the end of the file where they could be studied. Because each definition was safely contained in a macro, it was possible to write a detailed commentary of each routine without fear of affecting the final PostScript code in any way at all. Once this analysis was completed, it was possible to perform surgery on the offending PostScript definitions in an extremely controlled way. In particular, the FunnelWeb input file served as a repository for all the different versions of particular routines that were tried in order to get the definitions to work. A new (**Z**ero) macro was created for each version of each definition, and a commentary of how it performed added above it.

This case demonstrates that FunnelWeb is an extremely powerful tool for dissecting and documenting cryptic text files. Through the use of macros, particular parts of the file can be isolated and discussed without affecting the final product file in any way. In the example above, only a small part of the file was analysed, the rest being left as a blob, but in the general case, a cryptic text file could be inserted into FunnelWeb and then incrementally dissected (and possibly modified) until the result is a fully documented literate program. That this can be done without affecting the actual product file demonstrates the high degree of descriptive control that FunnelWeb provides.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

6.2 Making Ada ADTs More Abstract

Like many modern programming languages, Ada provides mechanisms for hiding information and structure. In particular, Ada provides a **package** facility that allows the programmer to declare objects in a package definition and define them in a corresponding package body. This works well for functions and procedures. However, in the case of types, implementation issues (in particular, the need to know the size of exported types) have led the designers of Ada to force the placement of private type definitions in the definition package rather than the implementation package. This means that some implementation details are present in the package definition for all to see. While not actually dangerous (the user of the package cannot make use of the information without recourse to "Chapter 13" of the Ada Language Reference Manual**[DOD83]**), this aspect of Ada is certainly unpleasant.

During the development of some Ada programs, FunnelWeb was used to solve this problem. Instead of creating a separate file for the package specification and package body, a single FunnelWeb file was created containing two sections, one for the each package part. The "private" part of the package specification was then moved (using a FunnelWeb macro definition) to the section describing the package body. Readers who wished only to read the package specification could read only the first part, which contained a fully documented description not containing the private definition.



6.3 Multiple Language Systems

With the prevalence of open systems and multi-vendor computing, it is often necessary to construct systems consisting of programs written in a number of different programming languages for a number of different systems. For example, a particular functionality might be implemented by a shellscript (invoked by the user) that calls a C program that makes a network connection to a Pascal program that queries a database. Quite often all these programs must conspire closely to execute their function. In the normal case, they must be written separately. FunnelWeb allows them to be written as a whole.

By creating a single FunnelWeb file that creates many product files in different languages, the programmer can describe the interaction between the different programs in any manner desired. Furthermore, because the different product files are all created in the same "text space" (i.e. in a single FunnelWeb file), it is easy for them to share information.

For example, in one real application FunnelWeb was used to create a system for printing files on a laser printer connected to a remote Vax Unix machine from a local Vax VMS machine. The system consisted of two files: a VMS DCL command procedure to run on the local node, and a Unix shellscript to run on the remote node. The user, by giving the print command, invoked the local VMS command procedure, which in turn fired up the remote Unix shellscript. The two scripts then cooperated to transfer the files to be printed and print them.

In addition to its usual documentation powers, FunnelWeb assisted in the creation of this system in two special ways. First, it allowed pieces of code from the two different command procedures to be partially interwoven in a description of their interaction. This is just not possible with comments. Second, it facilitated the use of shared information. For example, under some conditions, each file to be printed would be renamed and copied to the remote system using a particular constant filename (e.g. "`printfile.tmp`"). FunnelWeb allowed this constant filename to be included in a single macro definition which was invoked in the definition of each of the scripts. This ensured that the two scripts used the same name.

```
@A The following macro contains the temporary
file name used to allow the two shellscripts to
```

```
transfer each file to be printed.

@$@<printfile@>@M==@{printme.txt@}

@A Here are the scripts for the local VMS node
and the remote UNIX node.

@O@<vmscommandprocedure.com@>==@{@-
DCL commands
copy @<printfile@> unixnode::
DCL commands
@}

@O@<unixshellscript@>==@{@-
unix commands
print @<printfile@>
unix commands
@}
```

In the case of the printing system, the entire system was described and defined in a single FunnelWeb .fw file. In larger systems containing many FunnelWeb .fw files for many different modules in many different languages, the same trick can be pulled by placing FunnelWeb macro definitions for shared values into FunnelWeb include files. For example, a suite of implementations of network nodes, with each implementation being in a different programming language for a different target machine, could all share a table of configuration constants defined in macros in a FunnelWeb include file.

In summary, FunnelWeb's macro and include file mechanisms provide a simple way for programs written in different languages to share information. This reduces redundancy between the systems and hence the chance of inconsistencies arising.

6.4 The Case of the Small Function

Often, when programming, there is a need for a code abstraction facility that operates at the text level. If the statement "`a:=3;`" occurs often, it may be best simply to repeat it verbatim. If a sequence of one hundred statements is repeated often, it is normal to remove the code to a function and replace the occurrences by a function call. However, in between these two extremes are cases where a particular sequence of code is long enough and appears often enough to be troublesome, but which is bound so messily to its environment as to make a function call cumbersome.

For example, the following line of statements (referring to five variables declared *local* to a function) might appear ten times in a function:

```
a=b*3.14159; c=d % 256; e=e+1;
```

Now the "normal" rule of programming says that these statements should be placed in a procedure (also called a "function" in the C programming language used in this example), but here five local variables are used. Use of a procedure (function) would result in a procedure definition looking something like:

```
void frobit(a,b,c,d,e)
float *a,b;
int *c,d;
unsigned *e;
{*a=b << 8; *c=d % 256; *e=*e+1;}
```

and a procedure call something like

```
frobit(&a,b,&c,d,&e);
```

This might be workable in a language that allowed formal parameters to be specified to be bound only to particular variables. Similarly, it might be possible to avoid the parameter list in languages that support local procedures that can access non-local variables (such as Pascal). However, in our example here, in the C programming language, these options are not available, and so we must either create a function with five

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

parameters, or use the C macro preprocessor (the best solution). FunnelWeb provides the same macro facility for languages that do not have a built-in preprocessor.

In particularly speed-stressed applications, the programmer may be reluctant to remove code to a procedure because of the procedure-call overhead. FunnelWeb macros can help there too.

In summary, there sometimes arises in programming situations where the cost of defining a procedure is higher than the benefits it will bestow. Common reasons for this are the run-time procedure overhead and the messy binding problems caused by removing target code from its target context. FunnelWeb can help in these situations by allowing the programmer to define a text macro. This avoids all the problems and provides an additional incentive for the programmer to describe the piece of code so isolated.

6.5 When Comments are Bad

In the "good old days" of small machine memories and interpreted BASIC, programmers would eliminate the "REM" statements (comments) from their BASIC programs so as to save space and increase execution speed. Whilst this was obviously an appalling programming practice, the small memories and slow microprocessors often made this tempting, if not necessary.

Thankfully, times have changed since then, and most code is now compiled rather than interpreted. However, from time to time one still runs into an environment or situation, or special-purpose language, where comments are either unavailable (no comment feature) or undesirable. Here FunnelWeb can be used to fully document the code without resulting in any comments in the final code at all. For example:

- Comments in frequently used .h header files in C programs can have a significant impact on compilation speed. Often such header files are fairly cryptic and really ought to be well commented, but their authors are reluctant to.
- Comments are undesirable in PostScript header files that must be transferred repeatedly along communications channels (e.g. the Apple Macintosh LaserWriter header file).
- Interpreted programs in embedded systems.
- Hand written machine code in hex dump form could be commented.
- A programmer may wish to annotate a text data file containing lists of numbers that is to be fed into a statistical program that does not provide any comment facility for its input file.

In all these situations, FunnelWeb allows full integrated documentation without any impact on the final code.



Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

6.6 Documents That Share Text

FunnelWeb is very useful when preparing multiple documents that must share large slabs of identical text that are being constantly modified.

For example someone preparing two slightly different user manuals for two slightly different audiences might want the manuals to share large slabs of text, while still allowing differences between them. The following example shows how this can be done. The code is cluttered, but this clutter would not be a problem if the lumps of text were moderately large.

```
@O@<manual1.txt@>==@{@<M1@>@+@}
@O@<manual2.txt@>==@{@<M2@>@+@}

@$@<M1@>+=@{@<T1@>@}
@$@<M2@>+=@{@<T1@>@}
@$@<T1@>@M==@{@-
First lump of text shared by both documents.@+@}

@$@<M1@>+=@{Text for first document@+@}
@$@<M2@>+=@{Text for second document@+@}

@$@<M1@>+=@{@<T2@>@}
@$@<M2@>+=@{@<T2@>@}
@$@<T2@>@M==@{@-
Second lump of text shared by both documents.@+@}
```

An alternative approach, which might work better in situations where there are many small differences between the two documents rather than a few large ones, is to define a macro with two arguments, one for each product file document. Write the document from top to bottom, but place all stretches that differ between the two documents in a macro call.

```
@! Set the definition of @#D to
@!    @1 to create the shareholders report.
@!    @2 to create the customers report.
@$@#D@(@2@)@M==@{@1@}

@O@<report.txt@>==@{@-
ANNUAL REPORT TO @#D@(Shareholders@,Customers@)
================@#D@(============@,=========@)
This has been a very good year for The Very Big
Corporation of America. With your help, we have
```

```
been able to successfully
@#D@(@"screw the customers
          for every cent they have@"@,
     @"knock the shareholders into
        submission to bring you lower prices@"@).
With gross earnings approaching six trillion
dollars, we have been able to
@#D@(@"increase dividends@"@,
     @"lower prices@"@).
We expect to have an even better year next year.
@}
```

One application where text sharing can be particularly useful is in the preparation of computer documentation containing examples. For example, a book describing a new programming language might be full of examples of small programs written in the language which the user might want to try without having to type them all in. The "default" approach of keeping a copy of the examples in the text of the book and another copy in separate files is cumbersome and error prone, because both files have to be updated whenever an example is changed. A more sophisticated approach is to store each example in a separate file, and then use the "include file" facility of the word processor to include each example in the text. This is a better solution, but suffers from a few drawbacks. First, when editing the book in a word processor, the examples in the book will not be directly accessible or visible. To see an example, the writer would have to open the file containing the example in a separate window. This could become tedious if the text contained many examples, as many texts do. Furthermore, there is a risk that some example files will be included in the wrong place. Second, because the book is dependent on the included files, the book will end up consisting of a directory of a hundred or more files instead of just a few.

An alternative solution is to construct a single FunnelWeb .fw file that, when processed, produces both the book file and the example files. This solution assumes that the book consists of a text file containing commands for a typesetter such as TeX.

```
@O@<Book.tex@>==@{@#B@}

@$@#B+=@{@-
The first step to learning the object oriented
AdaCgol++ language is to examine a hello world
program.

\start{verbatim}
@<Ex1@>
\finish{verbatim}
```

```
@}

@$@<Ex1@>==@{@-
read iopack@+Enter !World~! !Hello~! ex pr flu X[1]@}
@O@<Ex1.c@>==@{@<Ex1@>@}

@$@#B+=@{@-
To understand the program, think of the execution
state as a plate of cheese...
@}
```

Most of the file will consist of part definitions of the additive macro @#B. The definition is "broken" to allow a macro definition, wherever an example appears.

The example above is a little messy because FunnelWeb does not allow macros connected to product files to be called, and it does not have text expressions that write to an product file as well as evaluating to text. Nevertheless, it presents a fairly clean solution to the problem of keeping the example programs in a computing text up to date.

6.7 Generics

It is well known that generics in programming languages are closely aligned with textual substitution. In fact, a good way to understand the generic facility of a new programming language is to ask oneself the question "In what way does this generic facility differ from simple text substitution?" The differences, if any, typically have to do with the difference in scoping between textual and intelligent substitution and whether the generic code is shared or copied by the implementation. In most cases the differences are quite minor.

Because generic facilities are so closely aligned with text substitution, it is possible to use FunnelWeb's parameterized macros to provide generics in programming languages that do not support generics. Simply write a FunnelWeb macro whose parameters are the parameters of the generic and whose body is the generic object.

The following FunnelWeb file gives an example of a fully worked Vax Pascal generic set package implemented using FunnelWeb parameterized macros. The package was written by Barry Dwyer of the Computer Science Department of the University of Adelaide in 1987 and was emailed to me on 11 November 1987. The generic package provides a set abstraction implemented using linked lists. Note the clever use of the instantiation parameters in type, function, and procedure names.

```
@$@<Generic Set Module@>@(@2@)==@{@-
@! @1 is the base type, @2 is the set type.
[inherit ('@1'), environment ('@2')]

module @2;

type  @2 = ^@2Record;
      @2Record = record
         Member: @1;
         Next: @2;
         end;

procedure Null@2 (var Result: @2);
begin new (Result);
Result^.Member := (- MaxInt)::@1;
Result^.Next := nil end;

function IsNull@2 (S: @2): boolean;
begin
IsNull@2 := S^.Member::integer = - MaxInt
```

```
end;

procedure ForEach@1 (S: @2; procedure DoIt (i: @1));
var   ThisS, NextS: @2;
begin ThisS := S;
while ThisS^.Member::integer <> - MaxInt do
   begin NextS := ThisS^.Next;
   DoIt (ThisS^.Member);
   ThisS := NextS end;
end;

function First@1 (S: @2): @1;
begin First@1 := S^.Member end;

function Is@1InSet (i: @1; S: @2): boolean;
   procedure TestEquals (j: @1);
   begin if Equal@1 (i, j) then Is@1InSet := true; end;
begin
Is@1InSet := false;
ForEach@1 (S, TestEquals);
end;

function Includes@2 (S1, S2: @2): boolean;
var Result: boolean;
   procedure TestIfInS1 (i: @1);
   begin
   if Result then
      if not Is@1InSet (i, S1) then
         Result := false;
   end;
begin Result := true;
ForEach@1 (S2, TestIfInS1);
Includes@2 := Result end;

function Disjoint@2s (S1, S2: @2): boolean;
var Result: boolean;
   procedure TestIfInS1 (i: @1);
   begin
   if Result then
      if Is@1InSet (i, S1) then
         Result := false;
   end;
begin Result := true;
ForEach@1 (S2, TestIfInS1);
Disjoint@2s := Result end;
```

```
function Equal@2 (S1, S2: @2): boolean;
begin
Equal@2 := Includes@2 (S1, S2) and
           Includes@2 (S2, S1);
end;

procedure Insert@1 (i: @1; var S: @2);
var   This, Pred, Succ: @2;
begin
if not Is@1InSet (i, S) then
   begin
   Pred := nil; Succ := S;
   while Succ^.Member::integer > i::integer do
      begin Pred := Succ; Succ := Succ^.Next end;
   if Succ^.Member::integer < i::integer then
      begin
      new (This);
      This^.Next := Succ;
      This^.Member := i;
      if Pred <> nil then
         Pred^.Next := This
      else
         S := This;
      end;
   end;
end;

procedure Insert@1s (S1: @2; var S2: @2);
var   This, Pred, Succ: @2;
   procedure Add@1 (i: @1);
   begin Insert@1 (i, S2) end;
begin
ForEach@1 (S1, Add@1);
end;

procedure Remove@1 (i: @1; var S: @2);
var   Pred, This: @2;
begin
Pred := nil; This := S;
while not Equal@1 (This^.Member, i) do begin
   Pred := This; This := This^.Next end;
if Pred <> nil then
   Pred^.Next := This^.Next
else
   S := This^.Next;
Dispose (This);
```

```
end;

procedure Dispose@2 (var S: @2);
var   Old: @2;
begin
while S <> nil do
   begin
   Old := S;
   S := S^.Next;
   Dispose (Old)
   end;
end;

end.
@}

@O@<NaryTreeSet.pas@>==@{@-
  @<Generic Set Module@>@-
@(@"NaryTree@"@,@"NaryTreeSet@"@)@}
@O@<NaryTreeSetSet.pas@>==@{@-
  @<Generic Set Module@>@-
@(@"NaryTreeSet@"@,@"NaryTreeSetSet@"@)@}
```

A great advantage of the approach reflected in the above example is that it allows the programmer to construct a generic object in a language that does not supply generics, *with complete typesafety*. This contrasts to the approach that might be used in a language such as C where the programmer might choose to construct a "generic" package by parameterizing a package with pointers to void. The resulting package is powerful but extremely untypesafe. Such a generic list package is used in the code of FunnelWeb itself and caused no end of problems, as the compiler had no way of telling if pointers to the correctly typed object were being handed to the correct list-object/function combination.

The major disadvantage of the text generic approach is that it causes the code of the generic object to be duplicated once for each instantiation. Depending on the number and size of the instantiations, this may or may not be acceptable.

Where the duplication of code is unacceptable, a hybrid approach may be taken. As in the C example, the programmer could write a single generic package using pointers to void or some other untypesafe mechanism. Then the programmer creates a FunnelWeb generic package whose functions do nothing more than call the functions of the untypesafe package, and whose types do nothing more than contain the types of the untypesafe package. This solution involves the use of untypesafe programming, but this is a one-off and if done carefully and correctly, the result can be a typesafe generic package involving minimal code duplication.

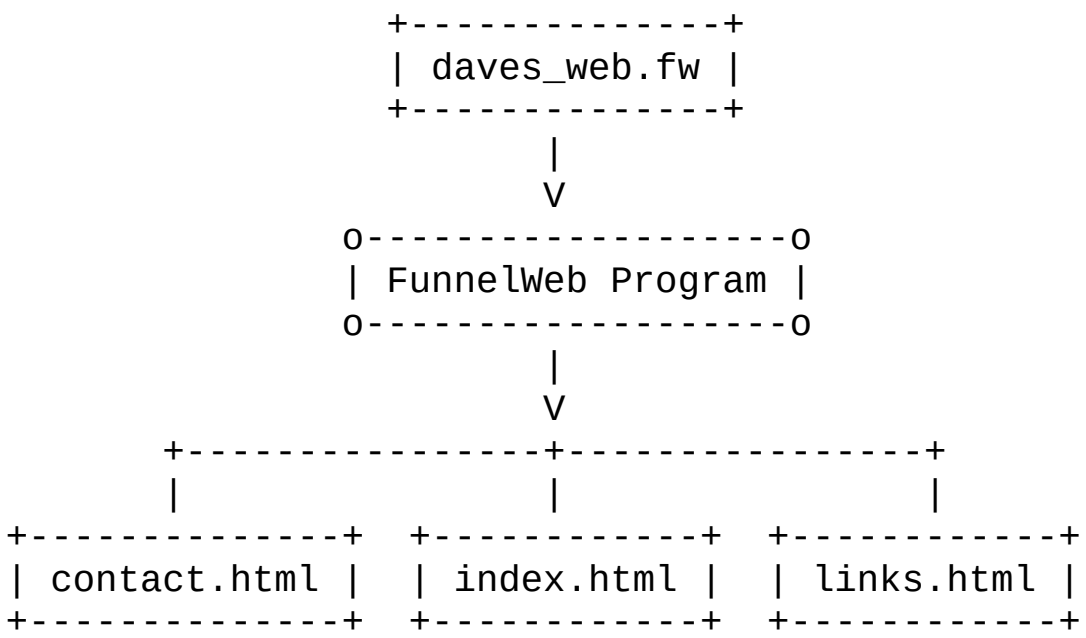
  

7 Making Webs With FunnelWeb

FunnelWeb was designed for the purposes of literate programming and was first created in 1986, long before the internet's World Wide Web appeared. So it came as a pleasant surprise to discover that FunnelWeb is an excellent tool for the preparation of websites! All the FunnelWeb webs, including the web you are reading right now, were created using FunnelWeb itself. This chapter explains how you can use FunnelWeb to make websites.

7.1 Introduction

How can FunnelWeb be used to make webs? Easy! Here's how it works. You create a single file called (say) daves_web.fw containing a Funnelweb output file macro (@O) for each page in the the web that you want to create. The daves_web.fw file contains the entire web. When you feed it through FunnelWeb, FunnelWeb generates all the HTML files in your web.



Here are some key points about this application:

- First, FunnelWeb can only generate the text files in your web (such as the .html files). You will still need to create the image files using the image manipulation tools you usually use; FunnelWeb does not help with the graphics, nor does it provide a GUI. You have to write raw HTML. However, FunnelWeb provides power tools for writing the HTML.

- Second, this is not a literate-programming application of webs. Here, FunnelWeb is being used purely as a macro preprocessor, and there will never be any cause to use FunnelWeb to generate documentation files (weaving). The files generated as above are FunnelWeb *output* files, not FunnelWeb HTML documentation produced by the FunnelWeb weaver. This is a raw macro preprocessor application.

So what benefit is there in converting one's web to a single FunnelWeb file? If you have just a small number of pages in your web (as in the

above diagram), there is probably not much benefit. However, if you have several pages in your web, FunnelWeb can provide huge benefits by enabling you to control the relationships between the various parts of your web and to parameterize it in whatever ways you want to. The result is a consistent style, a reduction of errors, and the power to execute broad ranging changes to the web with very little fuss. For example, FunnelWeb makes it easy to change the background of each page in your web, or to add a copyright notice at the bottom of every page.

The advantages of using FunnelWeb to make webs are as follows:

- Web is represented by a single file.
- No more multiple-file replace operations!
- Define a consistent style for the web.
- Parameterize the web. Make global changes easily.
- Use the power of HTML without the messy HTML syntax.
- Eliminates many syntax and spelling errors.
- Eliminates bad (internal) links.
- Eliminates form/CGI cross reference errors.
- A flexible text power tool always available.

The disadvantages of using FunnelWeb to make webs are as follows:

- No GUI interface. You have to write your web directly.
- It is more difficult to position graphics precisely.
- FunnelWeb files can look cryptic and messy.
- You have to run FunnelWeb between changing your web and viewing the changed pages.

Thus, if you make webs that are highly graphics intensive, or if you are uncertain about writing HTML, you should probably not use FunnelWeb. However, if you are generating large webs with many similar pages, FunnelWeb will eliminate much of the hassle in managing the whole complex. For large webs, this management capability is invaluable.

Whatever its advantages and disadvantages, FunnelWeb is certainly a **practical** production-quality webmaking tool. It has been used to make all of the webs in the following web spaces (each of which contains several subwebs):

Ross Williams
Rocksoft
Veracity

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

7.2 Getting Started

This page contains a brief hands-on tutorial to get you started in using FunnelWeb to make webs.

Hello Web

The best way to understand how FunnelWeb can help you make webs is to work through a simple example. Here is one:

```
@!************************************************

@O@<index.html@>@{
<HTML>

<HEAD>
<TITLE>Dave's Home Page</TITLE>
</HEAD>

<BODY BGCOLOR=#FFFFFF>
<P>Welcome to my home page. Check out my
<A HREF="links.html">links page</A>.
</BODY>

</HTML>
@}

@!************************************************

@O@<links.html@>@{
<HTML>

<HEAD>
<TITLE>Dave's Links</TITLE>
</HEAD>

<BODY BGCOLOR=#FFFFFF>
<P>Here are my links:

<P>
<A HREF="http://www.yahoo.com/">Yahoo</A><BR>
<A HREF="http://www.dilbert.com/">Dilbert</A><BR>
```

```
</BODY>

</HTML>
@}

@!************************************************
```

Cut and paste the above text into a file called dave.fw and run FunnelWeb on it by giving the command:

```
fw dave
```

Note: On my computer, when you cut and paste text from a web page, the pasted text contains a block of blanks before each line; you will have to delete these to make the example work, as FunnelWeb requires the first line of macro definitions to be in column one.

If all goes well, FunnelWeb should write out two files index.html and links.html. That's the generated web, and you should be able to browse it using your favorite web browser.

First Glimmers Of Usefulness

The sole benefit we have obtained so far is that the web is represented as a single file. This confers quite a few benefits on its own for medium-sized webs, such as the elimination of the need to perform multiple-file search and replace operations, but it's really only a start.

The next step is to recognise that the two pages above share quite a lot of text and that that text can be removed into FunnelWeb macros.

```
@!************************************************

@$@<Begin page@>@(@1@)@M@{
<HTML>
<HEAD>
<TITLE>@1</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
@}

@$@<End page@>@M@{
</BODY>
</HTML>
@}

@!************************************************
```

```
@O@<index.html@>@{

@<Begin page@>@(Dave's Home Page@)

<P>Welcome to my home page. Check out my
<A HREF="links.html">links page</A>.

@<End page@>
@}

@!************************************************

@O@<links.html@>@{
@<Begin page@>@(Dave's Links@)

<P>Here are my links:

<P>
<A HREF="http://www.yahoo.com/">Yahoo</A><BR>
<A HREF="http://www.dilbert.com/">Dilbert</A><BR>

@<End page@>
@}

@!************************************************
```

Now, although the file has actually got a little longer and looks far more complicated than it did, we've actually managed to extract two significant benefits from this reorganization.

> **Page style:** First, almost without even trying, we've created a page style. For example, to change the colour of all the pages in the web, we need just change the single body definition in the begin page macro. Similarly, if we wanted to add a copyright notice at the bottom of each page, we need change only the end page macro.
>
> **Page overhead:** Second, we've reduced the amount of text required to define each page. Instead of requiring ten lines of HTML to set up each page, we need just two FunnelWeb calls (two lines); the rest is text particular to the page being defined.

These advantages may seem minor, but when scaled up to a web containing a dozen or more pages, they become very significant.

The newfound power, in the case of the example, can be demonstrated by

changing the web from black on white to blue on white, and by adding another page:

```
@!************************************************

@$@<Begin page@>@(@1@)@M@{
<HTML>
<HEAD>
<TITLE>@1</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF TEXT=#000033>
@}

@$@<End page@>@m@{
</BODY>
</HTML>
@}

@!************************************************

@O@<index.html@>@{
@<Begin page@>@(Dave's Home Page@)

<P>Welcome to my home page. Check out my
<A HREF="links.html">links page</A> and my
<A HREF="hobbies.html">hobbies page</A>.

@<End page@>
@}

@!************************************************

@O@<links.html@>@{
@<Begin page@>@(Dave's Links@)

<P>Here are my links:

<P>
<A HREF="http://www.yahoo.com/">Yahoo</A><BR>
<A HREF="http://www.dilbert.com/">Dilbert</A><BR>

@<End page@>
@}

@!************************************************
```

```
@O@<hobbies.html@>@{
@<Begin page@>@(Dave's Hobbies@)

<P>This page is under construction!

@<End page@>
@}

@!**************************************************
```

Now that you have a Hello World file to play with, the remaining sections of this chapter will discuss the ways in which you can use the macro power now at your fingertips.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

7.3 Replacing Messy HTML Constructs

One of the greatest benefits of using FunnelWeb is its ability to replace messy slabs of HTML with a simple macro call. This page provides lots of examples of this.

Hyperlinks

Hyperlinks are messy and you can eliminate their messiness from your FunnelWeb web source by defining a macro for each link you want to use. For example:

```
@$@<Dilbert@>@Z@M@{@-
<A HREF="http://www.dilbert.com/">Dilbert</A>@}
@$@<Yahoo@>@Z@M@{@-
<A HREF="http://www.yahoo.com/">Yahoo</A>@}
```

While the definitions themselves are certainly messy, having made them, we are now free to refer to Dilbert and Yahoo within our webs in a very clean manner indeed. For example, the links page of the previous example now becomes just:

```
@O@<links.html@>@{
@<Begin page@>@(Dave's Links@)

<P>Check out @<Dilbert@> and @<Yahoo@>!

@<End page@>
@}
```

Images

Suppose we want to include a small image of a checkmark at various points without a web page. Here's the HTML that has to be included at each point:

```
<IMG SRC="bin/tick_red_12.gif"
WIDTH="12" HEIGHT="12"
HSPACE=4 VSPACE=0
ALT="*" BORDER="0">
```

Yuk! To eliminate this mess in our FunnelWeb/HTML, we can define a macro for a tick mark as follows:

```
@$@<Tick@>@M@{
<IMG SRC="bin/tick_red_12.gif"
WIDTH="12" HEIGHT="12"
HSPACE=4 VSPACE=0
ALT="*" BORDER="0">@}
```

Now, using tick marks without your text is easy. Here's an example:

```
@<Tick@>Low cost.<BR>
@<Tick@>Easy installation.<BR>
@<Tick@>Available now!<BR>
```

Large Form Fields

Another example is where you have form fields with lots of options. Here's an example of a commonly occurring form field that goes on for a couple of hundred lines!

```
<SELECT NAME="Country">
<OPTION>United States
<OPTION>Afghanistan
<OPTION>Albania
...
<OPTION>Zambia
<OPTION>Zimbabwe
</SELECT>
```

Using FunnelWeb, you can move all this to a single macro definition as follows:

```
@$@<Country form field@>@M@(
<SELECT NAME="Country">
<OPTION>United States
<OPTION>Afghanistan
<OPTION>Albania
...
<OPTION>Zambia
<OPTION>Zimbabwe
</SELECT>
```

Now, whenever you want a country field in a form, you can just write:

```
@<Country form field@>
```

Navigation Buttons

Sometimes the navigation constructs that appear within pages can become rather messy. Here's an example where we have three buttons at the bottom of each page.

```
<TABLE WIDTH="100%">
<TR>
<TD ALIGN="left"   VALIGN="bottom"><
 A HREF="links.html"><IMG SRC="bin/left.gif"></A></TD>
<TD ALIGN="center" VALIGN="bottom"><
 A HREF="home.html"><IMG SRC="bin/up.gif"></A></TD>
<TD ALIGN="right"  VALIGN="bottom"><
 A HREF="hobbies.html"><
 IMG SRC="bin/right.gif"></A></TD>
</TR>
</TABLE>
```

We can move all this to a single macro definition that has three parameters, one for each target page.

```
@$@<Nav@>@(@3@)@Z@M@{
@<P@>
<TABLE WIDTH="100%">
<TR>
<TD ALIGN="left"   VALIGN="bottom"><
 A HREF="@1"><IMG SRC="bin/left.gif"></A></TD>
<TD ALIGN="center" VALIGN="bottom"><
 A HREF="@2"><IMG SRC="bin/up.gif"></A></TD>
<TD ALIGN="right"  VALIGN="bottom"><
 A HREF="@3"><IMG SRC="bin/right.gif"></A></TD>
</TR>
</TABLE>
@}
```

Now, to add navigation buttons to a page, we can just write (near the end of the page).

```
@<Nav@>@(links.html@,home.html@,hobbies.html@)
```

If you wanted to, you could build the navigation buttons into the @<End page@> macro and set it up with three parameters instead.

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

7.4 Avoiding Errors And Inconsistencies

FunnelWeb provides the opportunity to eliminate all kinds of errors and inconsistencies by replacing text names with FunnelWeb macro names that cannot be misspelt without being detected by FunnelWeb. This page provides some examples of this.

Minor Style Elements

Have you ever done this:

```
This is <B Very Important</B>!
```

These kinds of errors can be eliminated by replacing HTML constructs, even simple ones, by FunnelWeb macros:

```
@$@<B@>@(@1@)@M@{<B>@1</B>@}
...
This is @<B@>@(Very Important@)!
```

While this is a bit messier, it eliminates the chance of an error because if you get the syntax wrong, FunnelWeb will generate an error at "compile time", which is a lot better than receiving email from some random web visitor. It's true that the same level of checking can be obtained by running an HTML syntax checker over your web. However, if you're using FunnelWeb for all its other benefits, you might as well use this aspect too.

File Names In Hyperlinks

Another source of error in webs is bad internal links. This can occur when you rename a file and forget to "fix up" all the links to it, and when you make a typing error when typing in a filename in a hyperlink.

You can use FunnelWeb to eliminate both of these kinds of error, by defining a macro for each output file. This is easier to do than explain:

```
@!**********************************************

@$@<Begin page@>@(@1@)@M@{
<HTML>
<HEAD>
```

```
<TITLE>@1</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF TEXT=#000033>
@}

@$@<End page@>@M@{
</BODY>
</HTML>
@}

@!************************************************

@O@<index.html@>@{
@<Begin page@>@(Dave's Home Page@)

<P>Welcome to my home page. Check out my
<A HREF="@<Links FILE@>">links page</A> and my
<A HREF="@<Hobbies FILE@>">hobbies page</A>.

@<End page@>
@}

@!************************************************

@$@<Links FILE@>@M@{links.html@}
@O@<links.html@>@{
@<Begin page@>@(Dave's Links@)

<P>Here are my links:

<P>
<A HREF="http://www.yahoo.com/">Yahoo</A><BR>
<A HREF="http://www.dilbert.com/">Dilbert</A><BR>

@<End page@>
@}

@!************************************************

@$@<Hobbies FILE@>@M@{hobbies.html@}
@O@<hobbies.html@>@{
@<Begin page@>@(Dave's Hobbies@)

<P>This page is under construction!

@<End page@>
```

```
@}

@!**********************************************
```

Again, this technique introduces a little messiness, but the benefits it brings are significant. Once you have deployed this technique throughout your web source file, you will then be free to change the names of any of your files without ever having to worry about fixing up any links - they will all be fixed automatically! Furthermore, if you delete an output file from your FunnelWeb source file, FunnelWeb will alert you to any "dangling links" because the references to the deleted page take the form of calls to a macro that is no longer defined! Of course, this only works if you don't mark your filename macros with @Z.

The only thing you have to get right is to ensure that the filename of the output macro is the same as the value of the corresponding naming macro. This isn't too hard to do because they are always on adjacent lines.

The benefit of eliminating all bad links from one's web is a benefit that can be obtained by running an external web analysis tool over your web. However, by using FunnelWeb file naming macros, you can be SURE that the web can never be generated with a bad link, whereas the web analysis tool only works if you remember to run it!

People's Names

Sometimes you might have to create a web that contains lots of long names that are hard to remember or spell. FunnelWeb can help here by eliminating the chance of error. Just define each name as macro and always call the macro when using the name. The benefit of doing this is that FunnelWeb will always detect a spelling error because if you make a spelling error in the macro name, FunnelWeb will generate a "macro not defined" error.

```
@$@<Marvin Hylrzbvwryvitz@>@M@{Marvin Hylrzbvwryvitz@}
```

If you want to, you could used a shortened form. For example:

```
@$@<Marvin@>@M@{Marvin Hylrzbvwryvitz@}
```

CGI Form Fields

One of the organizational challenges of implementing CGI form processing is getting all the field names right. Because (normally) the form is in one file, and the CGI script that processes it is in another file, it's very easy to accidentally use different names for the same field. For example, you might use email_adr in one and email-adr in the other.

FunnelWeb can be used to completely eliminate this problem as follows:

1. Put both the HTML file containing the form and the CGI file that processes it within the .fw file that embodies your web.

2. Define a macro for the name of each field and use the macro in both the form and the CGI script.

Here's an example:

```
@!************************************************

@$@<Email FIELD@>@M@{email_adr@}

@!************************************************

@O@<order.html@>@{
@<Begin page@>@(Order Form@)

<P>
<FORM ...>
<INPUT TYPE="text" NAME="@<Email FIELD@>">
...
</FORM>

@<End page@>
@}

@!************************************************

@O@<order.cgi@>@{@-
@<Begin CGI page@>
...
$v = $form{'@<Email FIELD@>'};
@<End CGI page@>
@}

@!************************************************
```

This technique more or less completely eliminates mismatch errors in form field names, for if any such error is made, FunnelWeb will generate a "macro not defined" error. All you have to do to benefit from this technique is get into the habit of using it.

When Global Replace Fails

Many people who write HTML directly use nerve wracking multi-file global replaces to perform the kind of web parameterization that FunnelWeb makes so easy. Unfortunately, these global replacements can go badly wrong because the target text may be used in many different contexts. In some cases, this approach fails completely.

For example, suppose that you have to create a hundred page web that has to be laced with two email addresses: a "business" email address and a "personal" email address. No problem so far. However, suppose that you haven't yet chosen a business domain name and, as a result, at the current time, the two email addresses are the same! If this were the case, you would have to put your address throughout all the pages and would LOSE the information of which were supposed to be business addresses and which were supposed to be personal ones.

FunnelWeb solves this problem (and others like it) completely by allowing you to define two macros one for each context. For example:

```
@$@<Business EMAIL@>@M@{dave@@someisp.com.au@}
@$@<Personal EMAIL@>@M@{dave@@someisp.com.au@}
```

These macros can then be used as appropriate. Later, when the domain name comes through and the business address changes, all that needs to be changed is a single definition:

```
@$@<Business EMAIL@>@M@{dave@@megacorp.com.au@}
@$@<Personal EMAIL@>@M@{dave@@someisp.com.au@}
```

There is no need to perform a multi-file global replace and evaluate the context of the use of each email address, as the information was not lost when the web was originally created, because of the two macros.

This example is just one of many similar situations that arise. By locating all the pages of your web in a single file and allowing you to define macros for different purposes, you can eliminate much of the time and danger of global replacements.

7.5 Defining A Consistent Style

We have already seen how FunnelWeb can be used, through the use of @<Begin page@> and @<End page@> macros to set up a particular page style throughout a web. However, this is really just the beginning of FunnelWeb's capacity to manage the style of a web. This page provides some other ideas.

Generic Style Macros

HTML contains some tags such as that have a high-level meaning ("emphasis") rather than a low-level meaning such as "italics". You can use FunnelWeb to create your own high level formatting macros so that you can later change the style of the web just by changing the macro definitions.

For example, suppose that your web contains lots of latin names of plants. You've decided that you want to highlight the names, but haven't decided how. Using FunnelWeb you can just define a macro:

```
@$@<Latin@>@(@1@)@M@{<B>@1</B>@}
```

This means that all latin names will be set in bold. However, if at a later date, you change your mind, you can just change the definition to some other format.

Margins

Here's another example. Suppose that you want some parts of the text in your web to be narrower than others. You can achive this by defining and using a macro like this:

```
@$@<Narrower@>@(@1@)@M@{@-
<TABLE><TR><TD WIDTH="200">
@1
</TD></TR></TABLE>
@}
```

If at a later date, you want to change all the widths, you just need to change the single definition.

7.6 Defining Macro Libraries

Separating Macros Into An Include File

Once you start using FunnelWeb to make webs, you'll discover that you find yourself defining and using the same macros over and over. Rather than redefining them for each new web you make, you can place them all in a FunnelWeb include file, so that they can be included in the FunnelWeb file for each web you make. Suppose that we create a file called style.fwi containing the following text:

```
@$@<Begin page@>@(@1@)@M@{
<HTML>
<HEAD>
<TITLE>@1</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF TEXT=#000033>
@}

@$@<End page@>@{
</BODY>
</HTML>
@}

@$@<Dilbert@>@Z@M@{@-
<A HREF="http://www.dilbert.com/">Dilbert</A>@}
@$@<Yahoo@>@Z@M@{@-
<A HREF="http://www.yahoo.com/">Yahoo</A>@}
```

Once this is done, the main web file can become:

```
@!************************************************

@i style

@!************************************************

@O@<index.html@>@{
@<Begin page@>@(Dave's Home Page@)

<P>Welcome to my home page. Check out my
```

```
<A HREF="@<Links FILE@>">links page</A>.

@<End page@>
@}

@!*************************************************

@$@<Links FILE@>@M@{links.html@}
@O@<links.html@>@{
@<Begin page@>@(Dave's Links@)

<P>Check out @<Yahoo@> and @<Dilbert@>.

@<End page@>
@}

@!*************************************************
```

Now we're starting to see some real simplification!

Note: It's important to attach @Z to any macro you place in an include file that you think might not be called in some webs. If you don't, FunnelWeb will issue "this macro is unused" errors for the included macros you don't use.

Library Macros

Once you've created one or more macro libraries in the form of FunnelWeb include files, you might find that sometimes you want to redefine some of the macros that the include file provides. For example, suppose that we are preparing an Australian web page and that when we refer to Yahoo, we want to refer to the Australian Yahoo site. If we try to redefine the Yahoo macro:

```
@i style

...

@$@<Yahoo@>@Z@M@{@-
<A HREF="http://www.yahoo.com.au/">Yahoo</A>@}
```

then FunnelWeb will issue an error "this macro is already defined". FunnelWeb does not allow macros to be redefined because this could be dangerous in a programming context. It would be very bad if a programmer thought they was invoking one piece of text when they were really invoking another. So FunnelWeb complains.

To stop FunnelWeb complaining, you can tag with a library macro marker @L any macro in the include file that you want to be able to redefine. For example:

```
@$@<Dilbert@>@Z@M@L@{@-
<A HREF="http://www.dilbert.com/">Dilbert</A>@}
@$@<Yahoo@>@Z@M@L@{@-
<A HREF="http://www.yahoo.com/">Yahoo</A>@}
```

This allows us to redefined the Yahoo and Dilbert macros.

Modifying Default Behaviour

The library macro mechanism allows you to set up all kinds of default behaviour that can be modified in specific webs. For example, suppose that we rewrote the macro library as:

```
@$@<Begin page@>@(@1@)@M@{
<HTML>
<HEAD>
<TITLE>@1</TITLE>
</HEAD>
@<Body@>
@}

@$@<End page@>@{
</BODY>
</HTML>
@}

@$@<Body@>@L@{@-
<BODY BGCOLOR=#FFFFFF TEXT=#000033>@}
```

Once this is done, we can modify the page background and colours of any web by redefining the Body macro without having to redefine the start and end of page macros.

```
@!************************************************

@i style

@$@<Body@>@{@-
<BODY BGCOLOR=#000000 TEXT=#FFFFFF>@}

@!************************************************
```

```
@O@<index.html@>@{
@<Begin page@>@(Dave's Home Page@)

<P>Welcome to my home page. Check out my
<A HREF="@<Links FILE@>">links page</A>.

@<End page@>
@}

@!************************************************

@$@<Links FILE@>@M@{links.html@}
@O@<links.html@>@{
@<Begin page@>@(Dave's Links@)

<P>Check out @<Yahoo@> and @<Dilbert@>.

@<End page@>
@}

@!************************************************
```

Use Of Additive Macros

Additive macros can be used to insert information into the middle of macros already defined. For example, suppose you defined:

```
@$@<Begin page@>@(@1@)@M@{
<HTML>
<HEAD>
<TITLE>@1</TITLE>
@<Header fields@>
</HEAD>
<BODY BGCOLOR=#FFFFFF TEXT=#000033>
@}

@$@<Header fields@>+=@{@}
```

Once the header fields macro is in place, you can add text such as style commands or keyword tags to each of the pages of your web without having to modify the header include file. Just add a definition, such as the following, to your main FunnelWeb file:

```
@$@<Header fields@>+=@{
```

```
<STYLE TYPE="text/css">
<!-- A {text-decoration: none} // -->
</STYLE>
@}
```

Selecting Absolute Or Relative Links

Sometimes it's important to generate a web that uses relative hyperlinks, and sometimes it's important to generate a web that uses absolute hyperlinks. For example, it's important to use relative links if you want to browse a web offline, or if you want to ensure that username and password fields in the URL are not obliterated by the following of an absolute link. Absolute links are important where a web that is normally adjacent to another web (and hence usually links to it using a relative link) is separated and must be browsed offline.

A good way to organize all this is to define FunnelWeb macros for the URLs of webs in include files, but allow them to be overruled by including files. For example, an file might contain the following definitions:

```
@$@<Rocksoft WWW@>@Z@M@L@{http://www.rocksoft.com/@}
@$@<Ross WWW@>@Z@M@L@{http://www.ross.net/@}
@$@<Veracity WWW@>@Z@M@L@{http://www.veracity.com/@}
```

These symbols are useful for referring to these various webs from within these webs. However, it is convenient for each subweb to refer to other subwebs using a relative path rather than an absolute path, so when generating (say) a subweb of the Ross webspace, the Ross macro could be redefined to:

```
@$@<Ross WWW@>@Z@M@{../@}
```

This technique can be used in various forms to manage the links in the online and offline versions of webs.

7.7 Parameterizing Entire Webs

The previous section has shown how individual macros within included files can be redefined to modify the details of a web. A more radical approach to web parameterization is to swap in entire include files!

Include Files As Parameter Sets

Suppose that we have a web that we are providing to three different web customers, where each web customer wants the web customized for their business. One way to approach this situation is to embody all the elements of the web that must be changed into macros located in an include file. Then define a different include file for each customer. So the include file for one customer could be:

```
@! Include file for MegaCorp
@$@<Home page title@>@{Welcome To MegaCorp@}
@$@<Background colour@>@{#FFFFFF@}
@$@<Emphasis@>@(@1@)@M@{<B>@1</B>@}
```

whereas the include file for a different customer might be:

```
@! Include file for MicroCorp
@$@<Home page title@>@{MicroCorp Home Page@}
@$@<Background colour@>@{#000000@}
@$@<Emphasis@>@(@1@)@M@{<I>@1</I>@}
```

In this way you can parameterize the style of an entire web, and generate different versions for different situations.

7.8 Hints And Conventions

Where To Include Include Files

It's better to include your include files at the end of the main web file, as if FunnelWeb generates an error, the line number in the listing file is easier to correlate with the main file if several files have not been included.

Separating Pages

It's a good idea to separate the macro for each page in a web in the FunnelWeb file, by a line of asterisks in a FunnelWeb comment. This makes the FunnelWeb source file easier to work on in your text editor.

```
@!************************************************

@O@<index.html@>@{
@<Begin page@>@(Dave's Home Page@)

<P>Welcome to my home page. Check out my
<A HREF="@<Links FILE@>">links page</A>.

@<End page@>
@}

@!************************************************

@$@<Links FILE@>@M@{links.html@}
@O@<links.html@>@{
@<Begin page@>@(Dave's Links@)

<P>Check out @<Yahoo@> and @<Dilbert@>.

@<End page@>
@}

@!************************************************
```

Input And Output Line Lengths

When using FunnelWeb to create webs, you will probably find that your input and output files have lines longer than the FunnelWeb standard 80 characters. So you may wish to include the following directives in your main and include files:

```
@p maximum_input_line_length = 200
@p maximum_output_line_length = 200
```

Macro Naming Conventions

Since 1994, I (Ross Williams) have been using FunnelWeb to generate all the webs in all my webspaces. During this time, I have developed some macro naming conventions which you may wish to adopt.

FILE: Use this suffix for macros that contain the names of files that form part of the web.

WWW: Use this suffix for macros that define web directory URLs, up to and including the trailing slash. The URL may be absolute or relative, depending on the context.

WWW/abs: Use this suffix where the URL must be absolute.

EMAIL: Use this suffix for email addresses.

FTP: Use this suffix for FTP directory addresses.

WINDOWNAME: Use this suffix for the names of browser windows.

Here are some examples:

```
@$@<Home FILE@>@Z@M@{index.html@}
@$@<Ross WWW@>@Z@M@{http://www.ross.net/@}
@$@<Ross WWW/abs@>@Z@M@{http://www.ross.net/@}
@$@<Ross EMAIL@>@Z@M@{ross@ross.net@}
@$@<Ross FTP@>@Z@M@{@-
ftp://www.ross.net/clients/ross/@}
@$@<Ross WINDOWNAME@>@Z@M@{ross@}
```

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

Reference

Developer

Tutorial
1 Introduction
2 Macros
3 Typesetting
4 Example
5 Hints
6 Examples
7 Webmaking

SEARCH

FunnelWeb Tutorial Manual

Copyright and Credits

Copyright Of The FunnelWeb Program

The FunnelWeb is made available under the GNU General Public Licence Version 2. See the FunnelWeb Developer Manual for a complete copy of this licence.

Copyright Of The FunnelWeb Webs

The entire contents of the following webs:

FunnelWeb
FunnelWeb Reference Manual
FunnelWeb Tutorial Manual
FunnelWeb Developer Manual

including, without limitation, all text, images, and sounds are copyright as follows:

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.
