

# INWEB

The Program

Chapter 1

Build 4/090319 Graham Nelson

# 1 Top Level

1/pc: *Program Control.w* The top level, which decides what is to be done and then carries this plan out.

1/cli: *Command Line and Errors.w* To parse the command line arguments with which inweb was called, and to handle any errors it needs to issue.

## *Purpose*

The top level, which decides what is to be done and then carries this plan out.

## *Definitions*

¶1. `inweb` is itself written as an `inweb` web. This hazardous circularity is traditional in literate-programming circles, much as people like to make compilers compile themselves: after all, if even the authors aren't willing to trust the code, why should anyone else? So this is a kind of demonstration of self-belief. `inweb` is probably fairly correct, because it is capable of tangling a program (`inweb`) which can then tangle a really complex and densely annotated program (Inform) which then passes over 1000 difficult test cases.

In my case, though, it was done simply because `inweb` had become that most evil of all system tools: a 4000-line Perl script resulting from bursts of careless invention every few months. When Woody Allen said that all literature is a footnote to Faust, he probably wasn't thinking of the Camel book, Perl's notorious manual, but he should have been. What can you say about a programming language where functions have no parameters but have to read them from an array named `$_`, where `local` defines a (partly) global variable, and where `my $a, $b` creates one local and one global, thanks to the comma being interpreted as marking a thrown-away evaluation of `my $a` (which, *obviously*, returns a value) and then an evaluation in void context of the nonexistent `$b`, which is helpfully and silently created as a global variable in the process? And yet, Perl is so quick... so easy... all that memory is yours at the slightest whim... and why work out how to derive data in format B from the data you already have in format A, when the devil on your shoulder whispers that you could just as easily store it both ways and save the trouble?

More seriously, Perl was used because it was the most standard scripting language with dynamic memory allocation available in 2003, when Inform 7 began: the largely preferable Python and Ruby had not then established themselves as fixtures.

¶2. In Perl floating-point arithmetic is used by default, since this is just what you probably wouldn't expect, and division in particular is slow in some Unix environments because it emulates floating point instructions rather than using hardware support: this was a nuisance when Inform was first ported to Unix boxes. So we gain some speed by writing what looks like a module inclusion, but is in fact a compiler `pragma`.

```
use integer;
```

¶3. Now we define the build of `Inweb`:

```
define $INWEB_BUILD "inweb [[Build Number]]"
```

¶4. As we will discover when we read it in, a web is either "chaptered" (its sections are distributed among Preliminaries, Chapter 1, ..., and some Appendices) or else "unchaptered", with all sections lumped together in "Sections".

```
$web_is_chaptered = 1;
```

¶5. `inweb` has a single fundamental mode of operation: on any given run, it is either tangling, weaving or analysing. These processes use the same input and parsing code, but then do very different things to produce their output, so the fork in the road is not met until halfway through `inweb`'s execution.

```
define $NO_MODE 0
define $ANALYSE_MODE 1
define $TANGLE_MODE 2
define $WEAVE_MODE 3
define $CREATE_MODE 4                a special mode for creating a new web, not acting on an existing one
$web_mode = $NO_MODE;                a value used to mean "not set yet"
```

¶6. It can also (in some functions, anyway) work on only selected portions of the web: such a selected piece is called a “target”, and its name is called a “sigil”. This is stored in the following textual variable, and can be a section name like `2/pine`, a chapter number like `12`, an appendix letter `A` or the preliminaries block `P`, or the special value `0` to mean the entire web.

```
$sigil_of_target = "0";                By default, the entire web is the target
```

¶7. In “swarm mode”, however, the user chooses a multiplicity of targets rather than just one, in which case the above value is meaningless.

```
define $SWARM_OFF 0
define $SWARM_INDEX 1                Make index(es) as if swarming, but don't actually swarm
define $SWARM_CHAPTERS 2            Swarm the chapters
define $SWARM_SECTIONS 3            That, and also all of the individual sections
$swarm_mode = $SWARM_OFF;
```

¶8. When tangling, we write a file whose name is:

```
$tangle_to = "";                This is set either by the command line or from the Contents
```

§1. This is the whole program in a nutshell, and it's a pretty old-school program: some input, some thinking, a choice of three forms of output. Perl scripts of course begin outside any function; this section is the only code at the top level in that way.

```
print "$INWEB_BUILD (Inform Tools Suite)\n";
make_command_line_settings();
if (($web_mode == $NO_MODE) || ($web_setting eq "")) <Show command-line usage and exit 2>;
read_configuration_file();
if ($create_setting ne "") <Create a new web 11>
else {
    <Read in the web 3>;
    <Display the statistics line 4>;
    if ($web_mode == $ANALYSE_MODE) <Analyse the web 5>;
    if ($web_mode == $TANGLE_MODE) <Tangle the web 6>;
    if ($web_mode == $WEAVE_MODE) <Weave the web 10>;
}
if ($no_inweb_errors == 0) { exit(0); } else { exit(1); }
```

§2. There's no sense writing out the whole manual; this usage note is intended for people who run across `inweb` and have no idea what it is.

(Show command-line usage and exit 2) ≡

```
print "[[Purpose]]\n";
print "Usage: inweb webname -action [-options] [target]\n";
print "  where 'webname' is a folder containing a web (an inweb project),\n";
print "  The most useful -action commands are:\n";
print "    -create: make a new web, creating its folder and contents\n";
print "    -tangle: make the program described in the web\n";
print "    -weave: make a human-readable booklet of the web\n";
print "  For options and less commonly used actions, see the inweb manual.\n";
exit(0);
```

This code is used in §1.

§3. `inweb` has basic support for a wider range of languages (the ones we need for the Inform project, really), but does detailed work only on C and on the Inform extension of C.

(Read in the web 3) ≡

```
read_literate_source();
language_set($bibliographic_data{"Language"});
if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
    create_base_types_hash();
}
parse_literate_source();
language_set($bibliographic_data{"Language"});
if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
    parse_C_like_sections();
}
```

This code is used in §1.

§4. To raise the morale of the user, really:

(Display the statistics line 4) ≡

```
print "\"\", $bibliographic_data{"Title"}, \" \" ";
if ($shared_structures+$private_structures > 0) {
    print $shared_structures+$private_structures, " structure(s)";
    if ($shared_structures > 0) { print ", $shared_structures shared"; }
    print ": ";
}
if ($no_chapters > 0) { print $no_chapters, " chapter(s) : "; }
print $no_sections, " section(s) : ", $no_paragraphs, " paragraph(s) : ",
    $no_lines, " line(s)\n";
```

This code is used in §1.

§5. “Analysis” invokes any combination of four diagnostic tools:

⟨Analyse the web 5⟩ ≡

```
if ($swarm_mode != $SWARM_OFF) {
    inweb_fatal_error("only specific parts of the web can be analysed");
}
if ($catalogue_switch == 1) { catalogue_the_sections($sigil_of_target, 0); }
if ($functions_switch == 1) { catalogue_the_sections($sigil_of_target, 1); }
if ($voids_switch == 1) { catalogue_void_pointers($sigil_of_target); }
if ($make_graphs_switch == 1) { compile_graphs($sigil_of_target); }
if ($scan_switch == 1) { scan_line_categories($sigil_of_target); }
```

This code is used in §1.

§6. We can tangle to any one of what might be several targets, numbered upwards from 0. Target 0 always exists, and is the main program forming the web. For many webs, this will in fact be the only target, but `inweb` also allows marked sections of a web to be independent targets – the idea here is to allow an Appendix in the web to contain a configuration file, or auxiliary program, needed for the main program to work; this might be written in a quite different language from the rest of the web, and tangles to a different output, but needs to be part of the web since it’s essential to an understanding of the whole system.

In this section we determine `$tn`, the target number wanted, and `$tangle_to`, the filename of the tangled code to write. This may have been set at the command line (in which case `$tangle_setting` contains the filename), but otherwise we impose a sensible choice based on the target.

⟨Tangle the web 6⟩ ≡

```
my $tn = 0;
if ($sigil_of_target eq "0") {
    ⟨Work out main tangle destination 7⟩;
} elsif (exists($sigil_section{$sigil_of_target})) {
    ⟨Work out an independent tangle destination, from one section of the web 8⟩;
} else {
    ⟨Work out an independent tangle destination, from one chapter of the web 9⟩;
}
if ($tangle_to eq "") { inweb_fatal_error("no tangle destination known"); }
$tangle_to = $web_setting."Tangled/".$tangle_to;
if ($tangle_setting ne "") { $tangle_to = $tangle_setting; }
tangle_source($tn, $tangle_to);
print "Tangled: $tangle_to\n";
```

This code is used in §1.

§7. Here the target number is 0, and the tangle is of the main part of the web, which for many small webs will be the entire thing.

⟨Work out main tangle destination 7⟩ ≡

```
$tn = 0;
if (exists $bibliographic_data{"Short Title"}) {
    $tangle_to = $bibliographic_data{"Short Title"};
} else {
    $tangle_to = $bibliographic_data{"Title"};
}
language_set($bibliographic_data{"Main Language"});
$tangle_to .= language_file_extension();
```

This code is used in §6.

§8. If someone tangles, say, 2/eg then the default filename is “Example Section”.

*(Work out an independent tangle destination, from one section of the web 8) ≡*

```
$tn = $section_tangle_target[$sigil_section{$sigil_of_target}];
if ($tn == 0) { inweb_fatal_error("section cannot be independently tangled"); }
$tangle_to = $section_leafname[$sigil_section{$sigil_of_target}];
```

This code is used in §6.

§9. If someone tangles, say, B then the default filename is “Appendix B”.

*(Work out an independent tangle destination, from one chapter of the web 9) ≡*

```
my $cn;
for ($cn=0; $cn<$no_chapters; $cn++) {
  if ($chapter_tangle_target[$cn] > 0) {
    if ($sigil_of_target eq $chapter_sigil[$cn]) {
      my $brief = $chapter_title[$cn];
      $brief =~ s/^.*?\:\s*//;
      $tangle_to = $brief;
      $tn = $chapter_tangle_target[$cn];
      last;
    }
  }
}
if ($tn == 0) {
  inweb_fatal_error("only the entire web, or specific sections, can be tangled");
}
```

This code is used in §6.

§10. Weaving is not actually easier, it’s just more thoroughly delegated:

*(Weave the web 10) ≡*

```
if ($swarm_mode == $SWARM_OFF) {
  my $shall_we_open = $open_pdf_switch;
  if ($shall_we_open == -1) { i.e., if it wasn't set at the command line
    if ($open_command_configuration ne "") { $shall_we_open = 1; }
    else { $shall_we_open = 0; }
  }
  weave_sigil($sigil_of_target, $shall_we_open);
} else { weave_swarm(); }
```

This code is used in §1.

§11. Lastly, here's a small utility for creating a new web – a slightly fiddly business, so just about worth automating.

⟨Create a new web 11⟩ ≡

```
system("mkdir -pv '$create_setting.'");
system("mkdir -pv '$create_setting./Figures'");
system("mkdir -pv '$create_setting./Materials'");
system("mkdir -pv '$create_setting./Sections'");
system("mkdir -pv '$create_setting./Tangled'");
system("mkdir -pv '$create_setting./Woven'");
system("cp -nv '$path_to_inweb_setting.inweb/Materials/Contents.w' '$create_setting.'");
system("cp -nv '$path_to_inweb_setting.inweb/Materials/Main.w' '$create_setting./Sections'");
```

This code is used in §1.

*Purpose*

To parse the command line arguments with which `inweb` was called, and to handle any errors it needs to issue.

---

1/cli. §1-8 Reading the command line; §9-10 The configuration file; §11-12 Error messages

---

*Definitions*

¶1. The command line options set the following variables. True/false options have `*_switch` variables, while textual ones are `*_setting`. (They also cause the two main program modes to be set: these are defined in Program Control.)

```

$analyse_structure_setting = "";
$catalogue_switch = 0;
$functions_switch = 0;
$convert_graphs_switch = 0;
$make_graphs_switch = 0;
$open_pdf_switch = -1;
$scan_switch = 0;
$tangle_setting = "";
$verbose_about_input_switch = 0;
$voids_switch = 0;
$web_setting = "";
$create_setting = "";
$only_setting = "";
$complete_PDF_leafname = "Complete.pdf";

```

*-analyse-structure: name of typedef struct to show usage of*  
*-catalogue: print catalogue of sections*  
*-functions: print catalogue of functions within sections*  
*-convert-graphs: run -make-graphs output through 'dot'*  
*-make-graphs: compile code to generate graphs*  
*-open-pdf: open any woven PDF in the OS once it is made*  
*-scan: simply show the syntactic scan of the source*  
*-tangle X: the pathname X, if supplied*  
*-verbose-about-input: print names of files read to stdout*  
*-voids: print void pointer usage*  
*set by -web: project folder relative to cwd*  
*set by -create: name of a new project to create*  
*set by -only: restrict a swarm to this chapter*  
*overridden when -only is used*

¶2. In order to run, `inweb` needs to know where it is installed – this enables it to find its configuration file, the macros file, and so on. Unless told otherwise on the command line, we'll assume `inweb` is present in the current working directory.

```
$path_to_inweb_setting = "";
```

¶3. The two external tools we may need to be present, beyond standard shell tools:

```

$pdftex_configuration = 'pdftex';
$dot_utility_configuration = 'dot';

```

¶4. We count the errors in order to be able to exit with a suitable exit code.

```
$no_inweb_errors = 0;
```

---

### §1. Reading the command line.

```
sub make_command_line_settings {
    my $i;
    my $targets = 0;
    ARGUMENT: for ($i=0; $i<=$#ARGV; $i++) {
        my $opt = $ARGV[$i];
        my $non_switch_follows = 0;
        if (($i < $#ARGV) && (not($ARGV[$i+1] =~ m/^-/))) { $non_switch_follows = 1; }
        if ($opt =~ m/^-/) {Parse this as a switch 2}
        else {
            if ($web_setting eq "") { $web_setting = $opt.'/' ; }
            else {Parse this as a target sigil 7};
        }
    }
}
```

### §2.

```
<Parse this as a switch 2> ≡
    $opt =~ s/^\-\-/\-/; allow a doubled-dash as equivalent to one
    if ($opt eq "-test-extensions") {
        print "(Test inweb's implementation of Inform's C extensions)\n";
        full_test_double_squares();
        exit(0);
    }
    if ($opt eq "-verbose-about-input") {
        $verbose_about_input_switch = 1; next ARGUMENT;
    }
    if ($opt eq "-at") {
        if ($non_switch_follows == 1) {
            $path_to_inweb_setting = $ARGV[$i+1]; $i++; next ARGUMENT;
        }
        inweb_fatal_error("-at must be followed by the pathname where inweb lives");
    }
    <Parse analysis options 3>;
    <Parse weaver options 4>;
    <Parse tangler options 5>;
    <Parse creation option 6>;
    inweb_fatal_error("unknown command line switch: $opt");
```

This code is used in §1.

## §3.

(Parse analysis options 3) ≡

```

if ($opt eq "-analyse-structure") {
    if ($non_switch_follows) {
        $analyse_structure_setting = $ARGV[$i+1]; $i++;
        enter_main_mode($ANALYSE_MODE);
        next ARGUMENT;
    }
    inweb_fatal_error("-analyse-structure must be followed by a structure name");
}
if ($opt eq "-catalogue") {
    $catalogue_switch = 1; enter_main_mode($ANALYSE_MODE); next ARGUMENT;
}
if ($opt eq "-functions") {
    $functions_switch = 1; enter_main_mode($ANALYSE_MODE); next ARGUMENT;
}
if ($opt eq "-voids") {
    $voids_switch = 1; enter_main_mode($ANALYSE_MODE); next ARGUMENT;
}
if ($opt eq "-make-graphs") {
    $make_graphs_switch = 1; enter_main_mode($ANALYSE_MODE); next ARGUMENT;
}
if ($opt eq "-convert-graphs") {
    $make_graphs_switch = 1; $convert_graphs_switch = 1;
    enter_main_mode($ANALYSE_MODE); next ARGUMENT;
}
if ($opt eq "-scan") {
    $scan_switch = 1; enter_main_mode($ANALYSE_MODE); next ARGUMENT;
}

```

This code is used in §2.

## §4.

(Parse weaver options 4) ≡

```

if ($opt eq "-weave") {
    enter_main_mode($WEAVE_MODE); next ARGUMENT;
}
if ($opt eq "-open") {
    $open_pdf_switch = 1; enter_main_mode($WEAVE_MODE); next ARGUMENT;
}
if ($opt eq "-closed") {
    $open_pdf_switch = 0; enter_main_mode($WEAVE_MODE); next ARGUMENT;
}
if ($opt eq "-only") {
    if ($non_switch_follows) {
        $only_setting = $ARGV[$i+1]; $i++;
        enter_main_mode($WEAVE_MODE);
        next ARGUMENT;
    }
    inweb_fatal_error("-only must be followed by a chapter number or appendix letter");
}

```

This code is used in §2.

## §5.

(Parse tangler options 5) ≡

```

if ($opt eq "-tangle") {
    enter_main_mode($TANGLE_MODE); next ARGUMENT;
}
if ($opt eq "-tangle-to") {
    if ($non_switch_follows) {
        $tangle_setting = $ARGV[$i+1]; $i++;
        enter_main_mode($TANGLE_MODE); next ARGUMENT;
    }
    inweb_fatal_error("-tangle-to must be followed by a filename to write");
}

```

This code is used in §2.

§6. The single creation option is an exception, since it doesn't act on an existing web:

(Parse creation option 6) ≡

```

if ($opt eq "-create") {
    if ($non_switch_follows == 1) {
        $create_setting = $ARGV[$i+1]; $web_setting = $create_setting; $i++;
        enter_main_mode($CREATE_MODE);
        next ARGUMENT;
    }
    inweb_fatal_error("-create must be followed by the pathname of a web");
}

```

This code is used in §2.

§7. A command-line argument not starting with a hyphen, and not already soaked up by a preceding argument such as `-tangle-to`, is a target sigil such as `2/eg` or `B`. Note that appendices are lettered `A` to `O`, but that `P` means the preliminary pages.

(Parse this as a target sigil 7) ≡

```

$targets++;
if ($targets > 1) { inweb_fatal_error("at most one target may be given"); }
$swarm_mode = $NO_SWARM;
if ($opt eq "index") {
    $swarm_mode = $SWARM_INDEX;
} elsif ($opt eq "chapters") {
    $swarm_mode = $SWARM_CHAPTERS;
} elsif ($opt eq "sections") {
    $swarm_mode = $SWARM_SECTIONS;
} elsif ($opt eq "all") {
    $sigil_of_target = "0";
} elsif ($opt =~ m/\//) {
    $sigil_of_target = $opt;
} elsif ($opt =~ m/^\d+$/) {
    $sigil_of_target = $opt;
} elsif ($opt =~ m/^[A-O]$/) {
    $sigil_of_target = $opt;
} elsif ($opt =~ m/^\P$/) {
    $sigil_of_target = $opt;
} else {

```

```

inweb_error("target not recognised: $opt");
print "The legal targets are:\n";
print "  all: complete web\n";
print "  P: all preliminaries\n";
print "  1: Chapter 1 (and so on)\n";
print "  A: Appendix A (and so on, up to Appendix O)\n";
print "  3/eg: section with abbreviated name \"3/eg\" (and so on)\n";
print "  index: HTML page indexing project\n";
print "  chapters: all individual chapters\n";
print "  sections: all individual sections\n";
exit(1);
}

```

This code is used in §1.

§8. We can only be in a single mode at a time:

```

sub enter_main_mode {
  my $new_mode = $_[0];
  if ($web_mode == $NO_MODE) { $web_mode = $new_mode; }
  if ($web_mode != $new_mode) {
    inweb_fatal_error("can only do one at a time - weaving, tangling or analysing");
  }
}

```

§9. **The configuration file.** `indoc` has only a tiny configuration file, mainly to point it to other tools it may need to use. Note that it needs none of these for tangling, so it doesn't actually matter if the settings are wrong in such a run.

```

sub read_configuration_file {
  my $cl;
  open(CONFIG, $path_to_inweb_setting.'inweb/Materials/inweb-configuration.txt')
  or die "inweb: can't open configuration file";
  while ($cl = <CONFIG>) {
    $cl =~ m/^\s*(.*?)\s*$/; $cl = $1;
    if ($cl =~ m/^\s*\/) { next; }
    if ($cl eq "") { next; }
    if ($cl =~ m/^\s*\s*\s*\s*(.*?)\s*\/) {
      my $setting = $1;
      my $value = $2;
      \(Make one of the configuration settings 10\);
    }
  }
  close CONFIG;
}

```

*skip comment lines*  
*skip blank lines*

§10. There's very little to see here:

(Make one of the configuration settings 10) ≡

```
if ($setting eq "pdftex") { $pdftex_configuration = $value; next; }
if ($setting eq "dot") { $dot_utility_configuration = $value; next; }
if ($setting eq "open-command") { $open_command_configuration = $value; next; }
inweb_error("inweb: bad configuration setting ($setting)");
```

This code is used in §9.

§11. **Error messages.** Ah, they kill you; or they don't.

```
sub inweb_fatal_error {
    my $message = $_[0];
    print STDERR "inweb: $message\n";
    exit(1);
}

sub inweb_error {
    my $message = $_[0];
    $no_inweb_errors++;
    print STDERR "inweb: $message\n";
}

sub inweb_error_at {
    my $message = $_[0];
    my $file = $_[1];
    my $line = $_[2];
    $no_inweb_errors++;
    print STDERR "inweb: $message\n";
    print STDERR " (" , $file, " line ", $line, ")\n";
}

sub inweb_error_at_program_line {
    my $message = $_[0];
    my $i = $_[1];
    my $sec = $line_sec[$i];
    inweb_error_at($message, $section_pathname_relative_to_web[$sec], $line_source_file_line[$i]);
}
```