# 2 Parsing a Web

`2/read`: *Reading Sections.w* To read the Contents section of the web, and through that each of the other sections in turn, and to collate all of this material into one big linear array of source-code lines.

`2/lcats`: *Line Categories.w* We are going to need to identify lines of source code as falling into 18 different categories – the start of a definition, a piece of a comment, and so on. In this section we define constants to enumerate these categories, and provide a debugging routine to show the classification we are using on the web we've just read.

`2/parse`: *The Parser.w* To work through the program read in, assigning each line its category, and noting down other useful information as we go.

`2/ident`: *Identifiers.w* To find the identifier names of functions and structures, and monitor in which sections of the program they are used; and so to police the accuracy of declarations at the head of each section.

# Reading Sections 2/read

*Purpose*

To read the Contents section of the web, and through that each of the other sections in turn, and to collate all of this material into one big linear array of source-code lines.

---



---

*Definitions*

**¶1.** This section describes a single, one-time event which happens early in every run of `inweb`: the reading in of the entire text of the web into memory, and the building of a suitable data structure to hold all this information, neatly docketed and reflecting its three layers – chapter, section, line.

It begins by reading the contents section, which really isn't a section at all (and perhaps we shouldn't pretend that it is by the use of the `.w` file extension, but we probably want it to have the same file extension, and its syntax is chosen so that syntax-colouring for regular sections doesn't make it look odd). When the word "section" is used in the `inweb` code, it almost always means "section other than the contents".

When the reading in is complete, the following variables have their final values:

```
$no_lines = 0;                                  Total lines in literate source, excluding contents
$no_sections = 0;                               Again, excluding contents: it will eventually be at least 1
$no_chapters = 0;                               Similarly, this will be at least 1
$no_tangle_targets = 0;                         And again
```

**¶2.** Once this phase is completed, the following arrays exist. We are pretty profligate with memory, but we can afford to be. (In the Parser section, we shall build even further arrays, but those aren't discussed here.)

For each individual chapter, numbered from 0 to `$no_chapters` minus 1 in order of declaration on the contents page, we store the following:

(C1) `chapter_sigil[]` is the sigil for the chapter owning this section: that is, the textual abbreviations used to identify chapters on the command line and elsewhere: `P` for Preliminaries, `7` for Chapter 7, `C` for Appendix C. In an apparently unchaptered project, the single chapter is called Sections and has sigil `S`: every section belongs to it.

(C2) `$chapter_title[]` is the title of the chapter, exactly as it is given in the Contents: for instance, "Chapter 3: Fresh Water Fish".

(C3) `$chapter_rubric[]` is the textual description of the chapter's purpose, as given in the Contents.

(C4) `$chapter_woven_pdf_leafname[]` is a leafname like `Appendix-A.pdf`, suitable for the woven PDF version of this individual chapter.

(C5) `$chapter_tangle_target[]` is the tangle ID number for tangling, or 0 if this is not marked for independent tangling.

**¶3.** Second, for each section other than the Contents (numbered 0 up to `$no_sections` minus 1, and in their canonical reading order, i.e., the order in which a human reader sees them in the typeset book):

(S1) `$section_chap[]` gives the chapter number to which the section belongs, which is between 0 and `$no_chapters` minus 1. (In a project without declared chapters, this will be 0, meaning the "Sections" pseudo-chapter.)

(S2) `$section_extent[]` is the number of lines in the section (which might be one more than the number of lines in its file, if it includes a chapter heading line).

(S3) `$section_pathname_relative_to_web[]` is the pathname of the section file within its own web folder.

(S4) `$section_leafname[]` is the leafname at the end of that. Obviously, it could be derived from (S3) without too much effort, but we often want it and it costs little to store, and since neither (S3) nor (S4) ever change we may as well keep both. It is also convenient to have a cross-referencing hash which lets us invert array number (S4): `$section_number_from_leafname{}`.

(ST) `$section_tangle_target[]` is the tangle ID number for the section – generally 0 to mean that it's part of the main tangle.

**¶4.** As this last implies, individual tangle targets are numbered from 0 up to `$no_tangle_targets` minus 1. Other than target 0, each target contains only a single section or a single chapter.

(A1) `$tangle_target_language[]` is the language of the tangle.

**¶5.** Lastly, for each individual line (numbered 0 up to `$no_lines` minus 1, and in their canonical reading order):

(L1) `$line_text[]` is the text as read in.

(L2) `$line_text_raw[]` is a duplicate for now. Later, the `$line_text[]` will be altered by parsing, whereas this won't, so it gives us access to the original form of the line.

(L3) `$line_sec[]` gives the section number (an index to the section arrays above) of the originating section. For the interleaved chapter heading lines placed between automatically chapters, the section number is the one just about to start, that is, the first section of the new chapter.

(L4) `$line_source_file_line[]` gives the line number, from 1, within the section file; or, for interleaved chapter headings, is 0.

---

§**1. Reading the contents page.** We read in the contents first, since that triggers everything else, by forcing reads of each section file in turn.

At the end, we have lines numbered 0 to `$no_lines-1` read in: this is the complete literate source of the web, so that we have the equivalent in memory of one long web file. Most of the lines come straight from the source files, but a few chapter heading lines are inserted if this is a multi-chapter web.

```
sub read_literate_source {
    read_contents_page("Contents.w");
}
```

§**2.** So here goes. The contents section has a syntax quite different from all other sections, and sets out bibliographic information about the web, the sections and their organisation, and so on.

```
sub read_contents_page {
    my $pathname_of_contents = $web_setting.$_[0];
    my $cline;                                          Line read in from the contents file
    my $clc = 0;                                        Line count within contents section (for use in error messages)
    my $scanning_bibliographic_block = 1;               Top bit, or bottom bit?
    my $scanning_chapter_purpose = 0;                   Reading the bit just after the new chapter?
    my $path_to_chapter_being_read = "";                Where sections in the current chapter live
    my $titling_line_to_insert = "";                    To be inserted automagically
    $bibliographic_data{"Declare Section Usage"} = "On";   The default
    $bibliographic_data{"Strict Usage Rules"} = "Off";     The default
    open CP, $pathname_of_contents
        or die "inweb: can't open contents section at: $pathname_of_contents\n";
    while ($cline = <CP>) {
        $cline =~ s/\s+$//; $clc++;
        if ($cline eq "") { $scanning_bibliographic_block = 0; next; }
        if ($scanning_bibliographic_block == 1) ⟨Read the bibliographic data block at the top 3⟩;
        ⟨Read the roster of sections at the bottom 7⟩;
    }
    close CP;
    if ($verbose_about_input_switch == 1) {
        print "Read contents section: '", $leafname, "' (", $cline, " lines)\n";
    }
    ⟨Check that the required bibliographic data was supplied 4⟩;
    ⟨Create the main tangle target 6⟩;
}
```

§**3.** The bibliographic data gives lines in any order specifying values of variables with fixed names; a blank line ends the block.

⟨Read the bibliographic data block at the top 3⟩ ≡

```
    if ($cline =~ m/^(.*?)\:\s*(.*?)\s*$/) {
        my $key = $1;
        my $value = $2;
        if ($key eq "License") { $key = "Licence"; }                    alias US to UK spelling
        if (($key eq "Title") || ($key eq "Short Title") || ($key eq "Author") ||
            ($key eq "Purpose") || ($key eq "Licence") ||
            ($key eq "Build Number") || ($key eq "Language") ||
            ($key eq "Index Extras") || ($key eq "Index Template") ||
            ($key eq "Cover Sheet") || ($key eq "Namespaces") ||
            ($key eq "Strict Usage Rules") || ($key eq "Declare Section Usage")) {
            $bibliographic_data{$key} = $value;
            if ((($key eq "Strict Usage Rules") || ($key eq "Declare Section Usage") ||
                ($key eq "Namespaces")) &&
                ($value ne "On") && ($value ne "Off")) {
                inweb_error_at("This setting must be 'On' or 'Off'", "Contents.w", $clc);
            }
        } else { inweb_error_at("no such bibliographic datum as '$key'", "Contents.w", $clc); }
    } else { inweb_error_at("expected 'Setting: Value' but found '$cline'", "Contents.w", $clc); }
    next;
```

This code is used in §2.

§**4.** Some bibliographic data settings are compulsory:

⟨Check that the required bibliographic data was supplied 4⟩ ≡
```
ensure_setting_of("Title"); ensure_setting_of("Author");
ensure_setting_of("Purpose"); ensure_setting_of("Language");
$bibliographic_data{"Inweb Build"} = $INWEB_BUILD;
$bibliographic_data{"Main Language"} = $bibliographic_data{"Language"};
```

This code is used in §2.

§**5.** Which requires the following:

```
sub ensure_setting_of {
    my $setting = $_[0];
    if (not (exists ($bibliographic_data{$setting}))) {
        inweb_fatal_error("The Contents.w section does not specify '$setting: ...'");
    }
}
```

§**6.** Tangle target 0 is the main program contained in the web we're reading; the programming language used by this will the one given by the `Language:` field in the contents section.

⟨Create the main tangle target 6⟩ ≡
```
$tangle_target_language[0] = $bibliographic_data{"Language"};
$no_tangle_targets++;
```

This code is used in §2.

§**7.** In the bulk of the contents, we find indented lines for sections and unindented ones for chapters.

⟨Read the roster of sections at the bottom 7⟩ ≡
```
$cline =~ m/^(\s*)(.*?)$/;                                  A pattern which cannot fail to match
my $whitespace = $1; local $title = $2;
if ($whitespace eq "") {
    if ($cline =~ m/^\"(.*)$/) { $scanning_chapter_purpose = 1; $cline = $1; }
    if ($scanning_chapter_purpose == 1) ⟨Record the purpose of the current chapter 8⟩
    else ⟨Read about a new chapter 9⟩;
} else ⟨Read about, and read in, a new section 12⟩;
next;
```

This code is used in §2.

§**8.** After a declared chapter heading, subsequent lines form its purpose, until we reach a closed quote: we then stop, but remove the quotation marks. Because we like a spoonful of syntactic sugar on our porridge, that's why.

⟨Record the purpose of the current chapter 8⟩ ≡
```
if ($cline =~ m/^(.*)\"\s*$/) { $cline = $1; $scanning_chapter_purpose = 0; }
if ($chapter_rubric[$no_chapters-1] ne "") {
    $chapter_rubric[$no_chapters-1] .= " ";
}
$chapter_rubric[$no_chapters-1] .= $cline;
next;
```

This code is used in §7.

§**9.** The title tells us everything we need to know about a chapter:

⟨Read about a new chapter 9⟩ ≡

```
my $new_chapter_sigil = "";                                   e.g., P, 1, 2, 3, A, B, ...
my $pdf_leafname = "";
my $ind_target = 0;
if ($title =~ m/^(.*?)\s*\(\s*Independent\s*(.*?)\s*\)\s*$/)
    ⟨Mark this chapter as an independent tangle target 10⟩;
if ($title eq "Sections") {
    $new_chapter_sigil = "S"; $path_to_chapter_being_read = "Sections/";
    $titling_line_to_insert = "";
    $pdf_leafname = "Sections.pdf";
    $web_is_chaptered = 0;
} elsif ($title eq "Preliminaries") {
    $new_chapter_sigil = "P"; $path_to_chapter_being_read = "Preliminaries/";
    $titling_line_to_insert = "";
    $pdf_leafname = "Preliminaries.pdf";
    $web_is_chaptered = 1;
} elsif ($title =~ m/^Chapter\s+(\d+)\:\s*(.*?)$/) {
    $new_chapter_sigil = $1; $path_to_chapter_being_read = "Chapter $1/";
    $titling_line_to_insert = $title.".";
    $pdf_leafname = "Chapter-$1.pdf";
    $web_is_chaptered = 1;
} elsif ($title =~ m/^Appendix\s+([A-O])\:\s*(.*?)$/) {
    $new_chapter_sigil = $1; $path_to_chapter_being_read = "Appendix $1/";
    $titling_line_to_insert = $title.".";
    $pdf_leafname = "Appendix-$1.pdf";
    $web_is_chaptered = 1;
} else {
    inweb_error_at("segment '$title' not understood", "Contents.w", $clc);
    print STDERR "(Must be 'Chapter <number>: Title', 'Appendix <letter A to O>: Title',\n";
    print STDERR "'Preliminaries' or 'Sections')\n";
}
⟨Create the new chapter with these details 11⟩;
```

This code is used in §7.

§**10.** A chapter whose title marks it as Independent becomes a new tangle target, with the same language as the main web unless stated otherwise.

⟨Mark this chapter as an independent tangle target 10⟩ ≡

```
$title = $1; $lang = $2;
$current_tangle_target = ++$no_tangle_targets;
$ind_target = $current_tangle_target;
$tangle_target_language[$no_tangle_targets] = $bibliographic_data{"Language"};
if ($lang ne "") { $tangle_target_language[$no_tangle_targets] = $lang; }
```

This code is used in §9.

§**11.**

⟨Create the new chapter with these details 11⟩ ≡

```
$chapter_sigil[$no_chapters] = $new_chapter_sigil;
$chapter_title[$no_chapters] = $title;
$chapter_rubric[$no_chapters] = "";
$chapter_tangle_target[$no_chapters] = $ind_target;
$chapter_woven_pdf_leafname[$no_chapters] = $pdf_leafname;
if ($ind_target == 0) { $current_tangle_target = 0; }
$no_chapters++;
```

This code is used in §9.

§**12.** That's enough on creating chapters. This is the more interesting business of registering a new section within a chapter – more interesting because we also read in and process its file.

⟨Read about, and read in, a new section 12⟩ ≡

```
my $source_file_extension = ".w";
if ($title =~ m/^(.*?)\s*\(\s*Independent\s*(.*?)\s*\)\s*$/) {
    ⟨Mark this section as an independent tangle target 13⟩;
} else {
    $section_tangle_target[$no_sections] = $current_tangle_target;
}
$section_chap[$no_sections] = $no_chapters - 1;
$section_extent[$no_sections] = 0;
my $path_to_section = $path_to_chapter_being_read.$title.$source_file_extension;
$path_to_section =~ s/ Template\.i6t$/\.i6t/;
$section_pathname_relative_to_web[$no_sections] = $path_to_section;
$section_leafname[$no_sections] = $path_to_section;
if ($section_leafname[$no_sections] =~ m/\/([^\/]*?)$/) {
    $section_leafname[$no_sections] = $1;
}
$section_number_from_leafname{$section_leafname[$no_sections]} = $no_sections;
read_file($path_to_section, $titling_line_to_insert, $no_sections);
$no_sections++;
```

This code is used in §7.

§**13.** Just as for chapters, but a section which is an independent target with language "Inform 6" is given the filename extension `.i6t` instead of `.w`. This is to conform with the naming convention used within Inform, where I6 template files – `inweb` files with language Inform 6 – are given the file extensions `.i6t`.

⟨Mark this section as an independent tangle target 13⟩ ≡

```
$title = $1; $lang = $2;
$section_tangle_target[$no_sections] = ++$no_tangle_targets;
$tangle_target_language[$no_tangle_targets] = $bibliographic_data{"Language"};
if ($lang ne "") {
    $tangle_target_language[$no_tangle_targets] = $lang;
    if ($lang eq "Inform 6") { $source_file_extension = ".i6t"; }
}
```

This code is used in §12.

**§14. Reading source files.** Note that we assume here that trailing whitespace on a line (up to but not including the line break) is not significant in the language being tangled for.

```
sub read_file {
    my $path_relative_to_web = $_[0];
    my $titling_line_for_this_chapter = $_[1];
    my $section_number = $_[2];
    my $file_line_count = 0;
    my $pathname = $web_setting.$path_relative_to_web;
    if (($titling_line_for_this_chapter ne "") &&
        ($titling_of_current_chapter ne $titling_line_for_this_chapter)) {
        $titling_of_current_chapter = $titling_line_for_this_chapter;
        $nl = '@*** '.$titling_of_current_chapter;
        ⟨Accept this as a line belonging to this section and chapter 15⟩;
    }
    open SECTIONF, $pathname or die "inweb: Unable to open $pathname\n";
    while ($nl = <SECTIONF>) {
        $file_line_count++;
        $nl =~ s/\s+$//;                                    remove trailing whitespace and the line break
        ⟨Accept this as a line belonging to this section and chapter 15⟩;
    }
    close SECTIONF;
    if ($verbose_about_input_switch == 1) {
        print "Read section: '", $pathname, "' (", $file_line_count, " lines)\n";
    }
}
```

**§15.**

⟨Accept this as a line belonging to this section and chapter 15⟩ ≡

*The text, with a spare copy protected from the parser's meddling:*

```
$line_text[$no_lines] = $nl;
$line_text_raw[$no_lines] = $nl;
```

*And where it occurs in the web:*

```
$line_sec[$no_lines] = $section_number;
$line_source_file_line[$no_lines] = $file_line_count;
```

*And keep count:*

```
$no_lines++;
$section_extent[$section_number]++;                    Not the same as the file line count!
```

This code is used in §14.

**§16.** A utility routine we need because the titling lines are special. Lines with a count of 1 begin their sections, obviously, but there are occasional lines with a count of 0 as well: these are the inserted chapter headings, and only occur in a chaptered web.

```
sub line_is_in_heading_position {
    my $i = $_[0];
    if (($line_source_file_line[$i] == 0) ||
        ($line_source_file_line[$i] == 1)) { return 1; }
    return 0;
}
```