The Swarm 3/swarm

Purpose

To feed multiple output requests to the weaver, and to present weaver results, and update indexes or contents pages.

§**1. Swarming.** Let us hope this is a glittering cloud, like the swarm of locusts in the title of Chapter 25 of Laura Ingalls Wilder's *On the Banks of Plum Creek*: at any rate it does make quite a deal of individual PDFs.

Each individual weave returns an ID number, its "weave number", which can be used to look up the individual results. This is stored in the arrays:

(C5) `$chapter_weave_number[]` is either the ID of the completed weave of this individual chapter, or -1 if no weave was made from it.

(S18) `$section_weave_number[]` is either the ID of the completed weave of this individual section, or -1 if no weave was made from it.

```
sub weave_swarm {
    my $i;
    for ($i=0; $i < $no_sections; $i++) { $section_weave_number[$j] = -1; }
    for ($i=0; $i < $no_chapters; $i++) { $chapter_weave_number[$j] = -1; }
    if ($swarm_mode >= $SWARM_SECTIONS) {
        for ($i=0; $i < $no_sections; $i++) {
            if (($only_setting eq "") || ($section_sigil[$i] =~ m/^$only_setting\//)) {
                $section_weave_number[$i] = weave_sigil($section_sigil[$i], 0);
            }
        }
    }
    if (($web_is_chaptered == 1) && ($swarm_mode >= $SWARM_CHAPTERS)) {
        for ($i=0; $i < $no_chapters; $i++) {
            if ($only_setting ne "") {
                if ($chapter_sigil[$i] ne $only_setting) { next; }
            }
            $chapter_weave_number[$i] = weave_sigil($chapter_sigil[$i], 0);
            if ($only_setting ne "") {
                $complete_web_weave_number = $chapter_weave_number[$i];
                $complete_PDF_leafname = $chapter_woven_pdf_leafname[$i];
            }
        }
    }
    if (($swarm_mode >= $SWARM_CHAPTERS) && ($only_setting eq "")) {
        $complete_web_weave_number = weave_sigil("0", 0);
    }
    weave_index_templates();
}
```

§**2.** After every swarm, we rebuild all the indexes specified in the Index Template list for the web:

```
sub weave_index_templates {
    ⟨Weave one or more index files 3⟩;
    ⟨Copy in one or more additional files to accompany the index 4⟩;
}
```

§**3.**

⟨Weave one or more index files 3⟩ ≡

```
    my $temp_list;
    my $leaf = "";
    if (exists ($bibliographic_data{"Index Template"})) {
        $temp_list = $bibliographic_data{"Index Template"};
    } else {
        if ($web_is_chaptered == 1) {
            $temp_list = $path_to_inweb_setting.'inweb/Materials/chaptered-index.html';
        } else {
            $temp_list = $path_to_inweb_setting.'inweb/Materials/unchaptered-index.html';
        }
        $leaf = "index.html";
    }
    while ($temp_list ne "") {
        my $this_temp;
        if ($temp_list =~ m/^(.*?)\s*\,\s*(.*)$/) {
            $temp_list = $2; $this_temp = $1;
        } else {
            $this_temp = $temp_list; $temp_list = "";
        }
        if ($leaf eq "") {
            $leaf = $this_temp;
            if ($leaf =~ m/^.*\/(.*?)$/) { $leaf = $1; }
        }
        print "Weaving index file: Woven/$leaf\n";
        weave_contents_from_template($this_temp, $leaf);
        $leaf = "";
    }
```

This code is used in §2.

§**4.** The idea here is that an HTML index may need some binary image files to go with it, for instance.

⟨Copy in one or more additional files to accompany the index 4⟩ ≡

```
my $copy_list = $path_to_inweb_setting.'inweb/Materials/download.gif'.
    ', '.$path_to_inweb_setting.'inweb/Materials/lemons.jpg';
if (exists ($bibliographic_data{"Index Extras"})) {
    $temp_list = $bibliographic_data{"Index Extras"};
}
my $path_to_copy_binaries = $web_setting."Woven";
while ($copy_list ne "") {
    my $this_file;
    if ($copy_list =~ m/^(.*?)\s*\,\s*(.*)$/) {
        $copy_list = $2; $this_file = $1;
    } else {
        $this_file = $copy_list; $copy_list = "";
    }
    my $leaf = $this_file;
    if ($leaf =~ m/^.*\/(.*?)$/) { $leaf = $1; }
    print "Copying additional index file: Woven/$leaf\n";
    system("cp '".$this_file."' '".$path_to_copy_binaries."'");
}
```

This code is used in §2.

§**5.** The following is where an individual weave task begins, whether it comes from the swarm, or has been specified at the command line (in which case the call comes from Program Control).

```
sub weave_sigil {
    my $request = $_[0];                                  The sigil of the target to be weaved
    my $open_afterwards = $_[1];                          Open the PDF in the host OS's viewer
    my $cover_sheet_flag = 0;                             Shall we have one?
    my $weave_section_match = '';                         Reg exp for sigil to match
    my $tex_file_leafname;                                What to call the resulting TeX file
    my $path_to_loom = $web_setting."Woven/";
    ⟨Translate the request sigil into details of what to weave 6⟩;
    weave_source($path_to_loom.$tex_file_leafname, $cover_sheet_flag, $weave_section_match);
    if ($lines_woven == 0) {
        inweb_fatal_error("empty weave request: $request");
    }
    my $wtn = run_woven_source_through_tex($path_to_loom.$tex_file_leafname,
        $open_afterwards, $cover_sheet_flag, $weave_section_match);
    ⟨Report on the outcome of the weave to the console 7⟩;
    return $wtn;
}
```

§**6.** From the sigil, we determine where to put the resulting TEX file, whether to make a PDF from it (always, at present), whether to open that PDF in the operating system (depends on the -open-pdf switch being used), whether to include a cover sheet (yes unless it's for just a single section), and which lines of the web should contribute to the source. We represent this last by supplying a regular expression which the sigil of the section owning the line should match: for instance, requiring this to match `11/` catches all of the lines in Chapter 11.

⟨Translate the request sigil into details of what to weave 6⟩ ≡

```
if ($request eq "0") {
    $weave_section_match = '.*';
    $booklet_title = "Complete Program";
    $tex_file_leafname = "Complete.tex";
    $cover_sheet_flag = 1;
} elsif ($request =~ m/^\d+$/) {
    my $cn = eval($request);
    $weave_section_match = '^'.$cn.'\/';
    $booklet_title = "Chapter ".$cn;
    $tex_file_leafname = "Chapter-".$cn.".tex";
    $cover_sheet_flag = 1;
} elsif ($request =~ m/^[A-O]$/) {
    my $cn = eval($request);
    $weave_section_match = '^'.$cn.'\/';
    $booklet_title = "Appendix ".$request;
    $tex_file_leafname = "Appendix-".$request.".tex";
    $cover_sheet_flag = 1;
} elsif ($request =~ m/^P$/) {
    my $cn = eval($request);
    $weave_section_match = '^'.$cn.'\/';
    $booklet_title = "Preliminaries";
    $tex_file_leafname = "Preliminaries.tex";
    $cover_sheet_flag = 1;
} elsif ($request =~ m/^(\S+?)\/(\S+)$/) {
    my $cn = eval($request);
    $weave_section_match = '^'.$1.'\/'.$2.'$';
    $booklet_title = $request;
    my $srequest = $request;
    $srequest =~ s/\//-/;
    $tex_file_leafname = $srequest.".tex";
    $cover_sheet_flag = 0;
} else {
    inweb_fatal_error("unknown weave request: $request");
}
```

This code is used in §5.

§**7.** Each weave results in a compressed one-line printed report:

⟨Report on the outcome of the weave to the console 7⟩ ≡

```
print "[", $request, ": ", $page_count[$wtn], "pp ",
    $pdf_size[$wtn]/1024, "K";
if ($overfull_hbox_count[$wtn] > 0) {
    print ", ", $overfull_hbox_count[$wtn], " overfull hbox(es)";
}
if ($tex_error_count[$wtn] > 0) {
    print ", ", $tex_error_count[$wtn], " error(s)";
}
print "]\n";
```

This code is used in §5.

§**8. Running TeX.** Although we are running `pdftex`, a modern variant of TEX, rather than the original, they are very similar as command-line tools; the difference is that the output is a PDF file rather than a DVI file, Knuth's original stab at the same basic idea.

In particular, we call it in "scrollmode" so that any errors whizz by rather than interrupting or halting the session. Because of that, we spool the output onto a console file which we can then read in and parse to find the number of errors actually generated. Prime among errors is the "overfull hbox error", a defect of TEX resulting from its inability to adjust letter spacing, so that it requires us to adjust the copy to fit the margins of the page properly. (In practice we get this here by having code lines which are too wide to display.)

```
sub run_woven_source_through_tex {
    my $tex_filename = $_[0];
    my $open_PDF = $_[1];
    my $with_cover_sheet = $_[2];
    my $weave_section_match = $_[3];

    my $path_to_tex = "";
    my $tex_leafname = "";
    my $console_filename = "";
    my $console_leafname = "";
    my $log_filename = "";
    my $pdf_filename = "";
    ⟨Work out these filenames and leafnames 9⟩;

    my $serious_error_count = 0;

    ⟨Call TeX and transcribe its output into a console file 10⟩;
    ⟨Read back the console file and parse it for error messages 11⟩;
    ⟨Remove the now redundant TeX, console and log files, to reduce clutter 12⟩;

    if ($open_PDF == 1) ⟨Try to open the PDF file in the host operating system 13⟩;
    return $no_weave_targets++;                    Return the weave ID number for this run of TEX
}
```

§**9.**

⟨Work out these filenames and leafnames 9⟩ ≡

```
$tex_leafname = $tex_filename;
if ($tex_leafname =~ m/^(.*)\/(.*?)$/) { $path_to_tex = $1; $tex_leafname = $2; }
$console_leafname = $tex_leafname; $console_filename = $tex_filename;
$console_leafname =~ s/\.tex$/.console/; $console_filename =~ s/\.tex$/.console/;
$log_filename = $tex_filename; $log_filename =~ s/\.tex$/.log/;
$pdf_filename = $tex_filename; $pdf_filename =~ s/\.tex$/.pdf/;
```

This code is used in §8.

§**10.**

⟨Call TeX and transcribe its output into a console file 10⟩ ≡

```
my $set_cwd = "";
if ($path_to_tex ne "") { $set_cwd = "cd \"".$path_to_tex."\"; "; }
$command = $set_cwd.$pdftex_configuration." -interaction=scrollmode \"".$tex_leafname."\" "
    .">\"".$console_leafname."\"";
system($command);
```

This code is used in §8.

§**11.** TEX helpfully reports the size and page count of what it produces, and we're not too proud to scrape that information out of the console file, besides the error messages (which begin with an exclamation mark in column 1).

⟨Read back the console file and parse it for error messages 11⟩ ≡

```
$overfull_hbox_count[$no_weave_targets] = 0;
$tex_error_count[$no_weave_targets] = 0;
$page_count[$no_weave_targets] = 0;
$pdf_size[$no_weave_targets] = 0;
open(CONSOLE, $console_filename) or die "no console file?";
while ($csl = <CONSOLE>) {
    if ($csl =~ m/Output written .*? \((\d+) page.*?(\d+) byte/) {
        $page_count[$no_weave_targets] = eval($1); $pdf_size[$no_weave_targets] = eval($2);
    }
    if ($csl =~ m/verfull \\hbox/) {
        $overfull_hbox_count[$no_weave_targets]++;
    } else {
        if ($csl =~ m/^\!/) {
            $tex_error_count[$no_weave_targets]++;
            $serious_error_count++;
        }
    }
}
close (CONSOLE);
```

This code is used in §8.

§**12.** The log file we never wanted, but TEX produced it anyway; it's really a verbose form of its console output. Now it can go. So can the console file and even the TEX source, since that was mechanically generated from the web, and so is of no lasting value. The one exception is that we keep the console file in the event of serious errors, since otherwise it's impossible for the user to find out what those errors were.

⟨Remove the now redundant TeX, console and log files, to reduce clutter 12⟩ ≡

```
if ($serious_error_count == 0) { system("rm \"".$console_filename."\""); }
system("rm \"".$log_filename."\"");
system("rm \"".$tex_filename."\"");
```

This code is used in §8.

§**13.** We often want to see the PDF immediately, so:

⟨Try to open the PDF file in the host operating system 13⟩ ≡

```
if ($open_command_configuration eq "") {
    inweb_error("no way to open PDF (see configuration file)", $pdf_filename);
} else {
    system($open_command_configuration." \"".$pdf_filename."\"");
}
```

This code is used in §8.

§**14. The Contents Interpreter.** This is a little meta-language all of its very own, with a stack for holding nested repeat loops, and a program counter and – well, and nothing else to speak of, in fact, except for the slightly unusual way that loop variables provide context by changing the subject of what is discussed rather than by being accessed directly.

define $TRACE_CI_EXECUTION 0 *For debugging*

```
sub weave_contents_from_template {
    my $path_to_template = $_[0];
    my $contents_page_leafname = $_[1];
    my $no_tlines = 0;
    ⟨Read in the source file containing the contents page template 15⟩;
    ⟨Open the contents page file to be constructed 16⟩;
    my $lpos = 0;                                   This is our program counter: a line number in the template
    $stack_pointer = 0;                             And this is our stack pointer for tracking of loops
    CYCLE: while ($lpos < $no_tlines) {
        my $tl = $tlines[$lpos++];                  Fetch the line at the program counter and advance
        $tl =~ m/^(.*?)\s*$/; $tl = $1;             Strip trailing spaces
        if ($TRACE_CI_EXECUTION == 1)
            ⟨Print line and contents of repeat stack 17⟩;
        if ($tl =~ m/^\s*\[\[(.*?)\]\]\s*$/) {
            my $command = $1;
            ⟨Deal with a Select command 18⟩;
            ⟨Deal with a Repeat command 19⟩;
            ⟨Deal with a Repeat End command 20⟩;
            print "Still here, ", $command, "\n";
        }
        ⟨Skip line if inside an empty loop 21⟩;
        ⟨Make substitutions of square-bracketed variables in line 24⟩;
        print CONTS $tl, "\n";                      Copy the now finished line to the output
    }
    close (CONTS);
}
```

§15. File handling.

⟨Read in the source file containing the contents page template 15⟩ ≡

```
if (not(open(TEMPL, $path_to_template))) {
    print "inweb: warning: unable to generate index because can't find template at ",
        $path_to_template, "\n";
    return;
}
while ($tl = <TEMPL>) {
    $tlines[$no_tlines++] = $tl;
}
close (TEMPL);
if ($TRACE_CI_EXECUTION == 1) {
    print "Read template <", $path_to_template, ">: ", $no_tlines, " line(s)\n";
}
```

This code is used in §14.

§16.

⟨Open the contents page file to be constructed 16⟩ ≡

```
my $path_to_contents = $web_setting."Woven/".$contents_page_leafname;
if (not(open(CONTS, '>'.$path_to_contents))) {
    print "inweb: warning: unable to generate index because can't open to write ",
        $path_to_contents, "\n";
    return;
}
```

This code is used in §14.

§17. The repeat stack and loops.

⟨Print line and contents of repeat stack 17⟩ ≡

```
my $j;
print sprintf("%04d: %tl\nStack:", $lpos-1, $tl);
for ($j=0; $j<$stack_pointer; $j++) {
    print " ", $j, ": ", $repeat_stack_variable[$j], "/", $repeat_stack_threshold[$j], " ",
        $repeat_stack_level[$j];
}
print "\n";
```

This code is used in §14.

§**18.** We start the direct commands with Select, which is implemented as a one-iteration loop in which the loop variable has the given section or chapter as its value during the sole iteration.

⟨Deal with a Select command 18⟩ ≡

```
if ($command =~ m/^Select (.*)$/) {
    my $sigil = $1;
    my $j;
    for ($j = 0; $j<$no_sections; $j++) {
        if ($section_sigil[$j] eq $sigil) {
            start_CI_loop("Section", $j, $j, $lpos);
            next CYCLE;
        }
    }
    for ($j = 0; $j<$no_chapters; $j++) {
        if ($chapter_sigil[$j] eq $sigil) {
            start_CI_loop("Chapter", $j, $j, $lpos);
            next CYCLE;
        }
    }
    inweb_error_at("don't recognise the chapter or section abbreviation $sigil",
        $path_to_template, $lpos);
    next;
}
```

This code is used in §14.

§**19.** Next, a genuine loop beginning:

⟨Deal with a Repeat command 19⟩ ≡

```
if ($command =~ m/^Repeat (Chapter|Section)$/) {
    my $from;
    my $to;
    my $lev = $1;
    if ($lev eq "Chapter") {
        $from = 0;
        $to = $no_chapters-1;
        if ($only_setting) {
            my $j;
            for ($j = 0; $j<$no_chapters; $j++) {
                if ($chapter_sigil[$j] eq $only_setting) {
                    start_CI_loop("Chapter", $j, $j, $lpos);
                    next CYCLE;
                }
            }
        }
    } elsif ($lev eq "Section") {
        my $within_chapter = heading_topmost_on_stack("Chapter");
        if ($within_chapter == -1) {
            $from = 0;
            $to = $no_sections-1;
        } else {
            $from = -1;
            my $sn;
            for ($sn = 0; $sn < $no_sections; $sn++) {
```

```
                if ($section_chap[$sn] == $within_chapter) {
                    if ($from == -1) { $from = $sn; }
                    $to = $sn;
                }
            }
        }
    } else {
        inweb_error_at("don't know how to repeat $lev: only Chapter or Section",
            $path_to_template, $lpos);
    }
    if ($from >= 0) { start_CI_loop($lev, $from, $to, $lpos); }
    next CYCLE;
}
```

This code is used in §14.

§**20.** And at the other bookend:

⟨Deal with a Repeat End command 20⟩ ≡

```
if ($command =~ m/^End (Repeat|Select)$/) {
    if ($stack_pointer <= 0) {
        inweb_error_at("stack underflow on contents template", $path_to_template, $lpos);
    }
    $repeat_stack_variable[$stack_pointer-1]++;
    if ($repeat_stack_variable[$stack_pointer-1] >=
        $repeat_stack_threshold[$stack_pointer-1]) {
        end_CI_loop();
    } else {
        $lpos = $repeat_stack_startpos[$stack_pointer-1];                    Back round loop
    }
    next CYCLE;
}
```

This code is used in §14.

§**21.** It can happen that a section loop, at least, is empty:

⟨Skip line if inside an empty loop 21⟩ ≡

```
my $rstl;
for ($rstl = $stack_pointer-1; $rstl >= 0; $rstl--) {
    if ($repeat_stack_variable[$stack_pointer-1] >=
        $repeat_stack_threshold[$stack_pointer-1]) { next CYCLE; }
}
```

This code is used in §14.

§**22.** If called with level `"Chapter"`, this returns the topmost chapter number on the stack; and similarly for `"Section"`.

```
sub heading_topmost_on_stack {
    my $level = $_[0];
    my $rstl;
    for ($rstl = $stack_pointer-1; $rstl >= 0; $rstl--) {
        if ($repeat_stack_level[$rstl] eq $level) {
            return $repeat_stack_variable[$rstl];
        }
    }
    return -1;
}
```

§**23.** This is the code for starting a loop, which stacks up the details, and similarly for ending it by popping them again:

```
sub start_CI_loop {
    my $level = $_[0];
    my $start_point = $_[1];
    my $end_point = $_[2];
    my $first_line_of_body = $_[3];
    $repeat_stack_level[$stack_pointer] = $level;
    $repeat_stack_variable[$stack_pointer] = $start_point;
    $repeat_stack_threshold[$stack_pointer] = $end_point+1;
    $repeat_stack_startpos[$stack_pointer++] = $first_line_of_body;
}
sub end_CI_loop {
    $stack_pointer--;
}
```

§**24. Variable substitutions.** We can now forget about this tiny stack machine: the one task left is to take a line from the template, and make substitutions of variables into its square-bracketed parts.

⟨Make substitutions of square-bracketed variables in line 24⟩ ≡

```
    while ($tl =~ m/^(.*?)\[\[(.*?)\]\](.*?)\s*$/) {
        my $left = $1; my $right = $3; my $subs = $2;
        if (exists($bibliographic_data{$subs})) {
            ⟨Substitute any bibliographic datum named 25⟩;
        } elsif ($subs =~ m/^(Chapter|Section|Complete) (.*)$/) {
            my $lev = $1;
            my $detail = $2;
            my $heading_number;
            if ($lev eq "Complete") {
                $heading_number = $complete_web_weave_number;
            } else {
                $heading_number = heading_topmost_on_stack($lev);
                if ($heading_number == -1) {
                    inweb_error_at("no $lev is currently selected",
                        $path_to_template, $lpos);
                }
            }
```

```
            if ($lev eq "Complete") {
                ⟨Substitute a detail about the complete PDF 26⟩;
            } elsif ($lev eq "Chapter") {
                ⟨Substitute a detail about the currently selected Chapter 27⟩;
            } else {
                ⟨Substitute a detail about the currently selected Section 28⟩;
            }
        } else {
            $subs = '<b>'.$subs.'</b>';
        }
        $tl = $left.$subs.$right;
    }
```

This code is used in §14.

§**25.** This is why, for instance, `[[Author]]` is replaced by the author's name:

⟨Substitute any bibliographic datum named 25⟩ ≡

```
    $subs = $bibliographic_data{$subs};
```

This code is used in §24.

§**26.** We store little about the complete-web-in-one-file PDF:

⟨Substitute a detail about the complete PDF 26⟩ ≡

```
    if ($detail eq "PDF Size") {
        $subs = ($pdf_size[$complete_web_weave_number]/1024)."KB";
    } elsif ($detail eq "Extent") {
        $subs = ($page_count[$complete_web_weave_number])."pp";
    } elsif ($detail eq "Leafname") {
        $subs = $complete_PDF_leafname;
    } else {
        $subs = $detail.' for complete web';
    }
```

This code is used in §24.

§**27.** And this is why [[Chapter Leafname]] turns into `$chapter_woven_pdf_leafname[$c]` for the current chapter number `$c`. Extent, PDF Size and Errors can only be determined if the chapter in question has already been woven on this run of `inweb`: hence the dashes if not.

⟨Substitute a detail about the currently selected Chapter 27⟩ ≡

```
    if ($detail eq "Title") {
        $subs = $chapter_title[$heading_number];
    } elsif ($detail eq "Purpose") {
        $subs = $chapter_rubric[$heading_number];
    } elsif ($detail eq "Leafname") {
        $subs = $chapter_woven_pdf_leafname[$heading_number];
    } elsif ($detail eq "Extent") {
        my $wtn = $chapter_weave_number[$heading_number];
        if ($wtn == -1) { $subs = "--"; } else {
            $subs = $page_count[$wtn]."pp";
        }
    } elsif ($detail eq "PDF Size") {
        my $wtn = $chapter_weave_number[$heading_number];
```

```
        if ($wtn == -1) { $subs = "--"; } else {
            $subs = ($pdf_size[$wtn]/1024)."KB";
        }
    } elsif ($detail eq "Errors") {
        my $wtn = $chapter_weave_number[$heading_number];
        if ($wtn == -1) { $subs = ""; } else {
            $subs = "";
            if ($overfull_hbox_count[$wtn] > 0) {
                $subs = $overfull_hbox_count[$wtn]." overfull hbox ";
            }
            if ($tex_error_count[$wtn] > 0) {
                $subs .= $tex_error_count[$wtn]." TeX error";
            }
        }
    } else {
        $subs = $detail.' for '.$lev.' '.$heading_number;
    }
```

This code is used in §24.

§**28.** And this, finally, is a very similar construction for Sections.

⟨Substitute a detail about the currently selected Section 28⟩ ≡

```
    if ($detail eq "Title") {
        $subs = $section_leafname[$heading_number];
        $subs =~ s/\.w$//;
    } elsif ($detail eq "Purpose") {
        $subs = $section_purpose[$heading_number];
    } elsif ($detail eq "Leafname") {
        $subs = $section_sigil[$heading_number];
        $subs =~ s/\//-/; $subs = $subs.'.pdf';
    } elsif ($detail eq "Code") {
        $subs = $section_sigil[$heading_number];
    } elsif ($detail eq "Lines") {
        $subs = $section_extent[$heading_number];
    } elsif ($detail eq "Source") {
        $subs = $section_pathname_relative_to_web[$heading_number];
    } elsif ($detail eq "Paragraphs") {
        $subs = $section_no_pars[$heading_number];
    } elsif ($detail eq "Mean") {
        my $denom = $section_no_pars[$heading_number];
        if ($denom == 0) { $denom = 1; }
        $subs = $section_extent[$heading_number]/$denom;
    } elsif ($detail eq "Extent") {
        my $wtn = $section_weave_number[$heading_number];
        if ($wtn == -1) { $subs = "--"; } else {
            $subs = $page_count[$wtn]."pp";
        }
    } elsif ($detail eq "PDF Size") {
        my $wtn = $section_weave_number[$heading_number];
        if ($wtn == -1) { $subs = "--"; } else {
            $subs = ($pdf_size[$wtn]/1024)."KB";
        }
    } elsif ($detail eq "Errors") {
```

```
        my $wtn = $section_weave_number[$heading_number];
        if ($wtn == -1) { $subs = ""; } else {
            $subs = "";
            if ($overfull_hbox_count[$wtn] > 0) {
                $subs = $overfull_hbox_count[$wtn]." overfull hbox ";
            }
            if ($tex_error_count[$wtn] > 0) {
                $subs .= $tex_error_count[$wtn]." TeX error";
            }
        }
    } else {
        $subs = $detail.' for '.$lev.' '.$heading_number;
    }
```

This code is used in §24.