The webfiles Package Version 0.5.39

Mark Potse *

1999/05/26

1 Introduction

The webfiles package makes it possible to plug the documentation of CWEB[1] and Spidery WEB programs into a LaTeX document. One can include any number of webs by including the weave processor's output with the command

\webfile[⟨*options*⟩]{⟨*filename*⟩}

where ⟨*filename*⟩ is the name of the .tex file (with or without the extension) that is output by the weave processor and ⟨*options*⟩ is a comma-separated list of options, as explained later. Of course, the webfiles package must be loaded:

\usepackage[⟨*options*⟩]{webfiles}

in the preamble of the document. The package recognizes the same options as the \webfiles command.

This manual does not describe the usage of WEB systems themselves; they have their own manuals[1, 2, 3]. In what follows, it is assumed that you read at least one of them, but note that some of the information here overrides what was said there.

This package is based on the cweb style by Joachim Schrod (even this manual is based on his). Apart from the fact that, with this package, web's are 'plugged in' while in Schrod's style they are the main and only body, there is a number of differences:

1. This package can also work with Spidery webs.
2. The layout (determined by the document class and maybe other packages) is not changed.
3. Existing counters are not affected.
4. The section numbers in weave's output are used.
5. The contents of all webs are listed in a "List of Programs", if the user wishes so.
6. The user may choose if pagebreaks may occur inside sections. If not, the result is a 'ragged bottom'.
7. A section is only printed if it is 'on'. (Refer to the documentation of a WEB to see what 'on' means in this context.)
8. The multicol package is used, if present, to typeset the identifier index in two columns. Therefore a pagebreak between the identifier index and the list of refinements is no longer necessary. If multicol cannot be found, LaTeX's \twocolumn is used.
9. The index and list of refinements can be suppressed in the LaTeX document.
10. Format (@f) statements can be suppressed in the LaTeX document.
11. Support for hyperlinks is included, as well as direct support for PDFTeX.

* Medical Physics Department, University of Amsterdam.
Email: M.Potse@amc.uva.nl

[1] version 3, this package does not work with older versions

1.1 The overall structure

The philosophy behind this package is that you, like it's author, may wish to import more than one WEB program, maybe in more than one programming language, in a single LaTeX document. With this package, you can regard to the `.web` file as just a part of your document, like the parts that you `\include` when writing a large document in more than one file, and in the documentation parts of the WEB program you can use all features of LaTeX the way you would in that case.

CWEB Version 3.0 introduced the notion of a starred section's "depth." As in the layout produced by `cwebmac.tex` the depth value does only influence the web's table of contents (and "List of Programs"), not the rendering of the title within the text. (`cwebmac.tex` is the original, plain-based macro package used to typeset CWEB sources; I refer henceforth to it as the "plain version.")

The *document* structure is tagged in the mother document, with `\section` and such. Although it is not forbidden to use sectioning commands in the documentation part of the WEB program, it doesn't seem useful to do so. The only place where a sectioning command could be useful is in the limbo section.

The 'mother document' must load the "`webfiles`" package:

```
\usepackage{webfiles}
```

Then you can include a WEB program with the following command:

`\webfile{`⟨*filename*⟩`}`

where ⟨*filename*⟩ is the `.tex` file that is output by a `weave` processor.

The resulting structure of the mother document is exemplified in figure 1. Text may be inserted anywhere inbetween the tags shown there.

2 Customization

Both the `\webfile` command and the `\usepackage{webfiles}` command accept an optional argument that may contain a comma-separated list of options. Options specified to `\usepackage` act as global defaults that can be overridden in the optional argument to `\webfile`. Therefore each option has an opposite. The recognized options are summarized in table 1, and discussed in the following sections. Other customizations are possible through the redefinition of macros and the setting of counters; the most important of these will be explained in this section.

2.1 The index and the list of refinements

An identifier index and a list of refinements are created at the end of each WEB document, if you didn't call `weave` with the `-x` option[2]. You can specify an introductionary text for the index with the tag `\xwebIndexIntro`, the introduction is the argument of this tag. To suppress generation of an identifier index, even when `weave` was not called with the `-x` option you can use an optional argument to the `\webfile` command; for example:

```
\webfile[noindex]{foo}
```

To suppress identifier indexes in all web's, use this option in the `\usepackage` command:

```
\usepackage[noindex]{webfiles}
```

You can use the `index` option in the `\webfile` command to override this global default.

The same applies to the list of refinements, using the options `reflist` and `noreflist`.

[2]The '`-x`' option suppresses output of the index and reflist—see the documentation.

```
\documentclass{report}
\usepackage{webfiles}

\begin{document}

\title{Three solutions for the Travelling Salesman problem}
\author{Joe L. User}

\maketitle

\tableofcontents    % optional
\listoffigures      % etc., optional
\listofprograms     % optional

\chapter{A PSPACE solution for the Travelling Salesman}
\webfile{pspace}

\chapter{The Travelling Salesman problem, solved by Simulated
  Annealing}
\webfile{anneal}

\chapter{A Neural Network solution for the Travelling Salesman}
\webfile{neu_net}

\appendix
\chapter{The Travelling Salesman Problem, solved by telepresence}
\input{internet-user}

\end{document}
```

Figure 1: Exemplified document structure

2.2 The table of contents

Each web document gets its own table of contents. It can be suppressed with the `nocon` option; the opposite option is `contents`. You can produce the table of contents of any web document in any position with the `\xwebContents` command. This command takes one argument: the basename of the web document, and does not work with Spidery webs (yet[3]).

The tables of contents are produced like all LaTeX's tables of contents etc., in the second LaTeX run. They are stored in a file with name ⟨*basename*⟩`.con`, where ⟨*basename*⟩ is the basename of the web document.

2.3 The List of Programs

In addition to the table of contents there is a "List of Programs", analogous to the "List of Figures" etc. that standard LaTeX provides. The list can be produced with a

```
\listofprograms
```

command, which can be placed at any position. It is customary to put it at the beginning of the mother document, after it's table of contents. As with the List of Figures, the List of Programs is generated at the second LaTeX run. It can list programs, main sections and normal sections; the default is to list only programs and main sections. You can control what goes into the list by setting the counter `xwebLopDepth`:

[3]I consider releasing a revised version of Spider that is more compatible with CWEB, and has less bugs

option	*opposite*	*effect*
`index`*	`noindex`	print identifier index
`noindex`	`index`	don't print identifier index
`reflist`*	`noreflist`	print List of Refinements
`noreflist`	`reflist`	don't print List of Refinements
`contents`*	`nocon`	print table of contents
`nocon`	`contents`	don't print table of contents
`allsections`*	`onlychanges`	print all sections
`onlychanges`	`allsections`	print only changed sections
`showformats`*	`hideformats`	show `@f` declarations
`hideformats`	`showformats`	don't show `@f` declarations
`raggedbottom`*	`flushbottom`	try not to break modules across pages
`flushbottom`	`raggedbottom`	try to create a flush bottom, break modules if necessary
`hyperref`	`nohype`	make hyperlinks using `hyperref` package
`pdftex`	`nohype`	make hyperlinks and PDF outlines with PDFTEX
`nohype`*	...	no hyperlinks
`nofiles`	`writefiles`	dont't write a contents file (equivalent to setting `\setcounter{xwebLopDepth}{-9}`).
`writefiles`*	`nofiles`	do write a contents file. (equivalent to setting `\setcounter{xwebLopDepth}{9}`).

Table 1: Webfiles options. Defaults are tagged with a *.

- Normal sections get into it if `xwebLopDepth` is greater than or equal to 10.
- All starred sections that have a depth smaller than `xwebLopDepth` will be mentioned.
- Programs will be mentioned unless `xwebLopDepth` is smaller than or equal to −10.

By setting `xwebLopDepth` inbetween `\webfile` commands you can handle each `WEB` file differently, if you wish. The default value is 9. It is a LaTeX counter, so its value can be changed using the `\setcounter` and `\addtocounter` commands.

The PDF outline (if you use PDFTEX) can be controlled in the same way, using the counter `xwebOutlineDepth`, except that normal sections never get into the outline.

The entries for the List of Programs are put into the file "⟨*jobname*⟩`.lop`", where ⟨*jobname*⟩ is the jobname that TEX chose for the mother document (usually the name of the `.tex` file, without the extension). Edit this file only if you're desparate, as with the `.toc` and `.lof` files etc.

2.4 Newpages

The default behaviour of the `webfiles` package is never to break sections across pages if they fit on a single page. If sections may be broken across pages, the `webfiles` package is able create 'flush bottom' pages. To enable this, specify the `flushbottom` option to the `\webfile` or the `\usepackage` command. To re-enable the default behaviour locally, specify the `raggedbottom` option.

A main section can start a new page regardless if it will fit on the current one. To change this behaviour, set the counter `xwebSecNoEject`, which keeps the lowest group level where no new page is started at a main section. The default is 3. It is a LaTeX counter, so its value can be changed using the `\setcounter` and `\addtocounter` commands.

2.5 Hyperlinks

If the `hyperref` option is specified, all module references are turned into hyperlinks using the `hyperref` package[4]. To use this feature, you have to import the `hyperref` package yourself, before importing the `webfiles` package, e.g.:

```
\usepackage[hypertex]{hyperref}
```

```
\usepackage[hyperref]{webfiles}
```

Alternatively, with the `pdftex` option, direct support for PDFTEX can be enabled. This provides the same look and feel as the `pdfcwebmac` macros, including the 'PDF Outline', but of course works only with PDFTEX.

Both types of hyperlinked references can be switched off with the `nohype` option.

2.6 Output in other languages

`WEB` documentation can contain a few—predefined—informal texts. As in the plain version, you can change these texts, for example to obtain output in an other language than English, by redefining some macros. The macro names concerned are listed below, together with their default definitions.

```
\def\xwebIndexName{Index of \xwebJobname}
\def\xwebReflistName{List of Refinements in \xwebJobname}
\def\xwebCRAlso{\xwebCrossRef{See also section}}
\def\xwebCRsAlso{\xwebCrossRef{See also sections}}
\def\xwebCRCite{\xwebCrossRef{This code is cited in section}}
\def\xwebCRsCite{\xwebCrossRef{This code is cited in sections}}
\def\xwebRLCite{\xwebCrossRef{Cited in section}}
\def\xwebRLsCite{\xwebCrossRef{Cited in sections}}
\def\xwebCRUse{\xwebCrossRef{This code is used in section}}
\def\xwebCRsUse{\xwebCrossRef{This code is used in sections}}
\def\xwebRLUse{\xwebCrossRef{Used in section}}
\def\xwebRLsUse{\xwebCrossRef{Used in sections}}
\def\xwebCRChanged{%
   \xwebCrossRef{The following sections were changed by the change file:}%
   \let\*\relax}
\def\xwebCREt{ and~}
\def\xwebCRsEt{, and~}
\def\xwebLopName{List of Programs}
\def\xwebTocName{Contents of \xwebJobname}
\def\xwebRefMacrosHere{Preprocessor Definitions}
\def\xwebSectionName{Section}
\def\xwebPageName{Page}
```

2.7 Documenting changefiles

If a web file is included that has some sections changed by a changefile you can have the `\webfile` command print only the changed sections by giving the `onlychanges` option. As with all options, this option can also be given to the `\usepackage` command. Its opposite option is `allsections`. Specifying `onlychanges` is equivalent to saying `\let\maybe=\iffalse` in the plain version.

2.8 Miscellaneous

You can control the appearance of web section numbers by redefining the appearance of `\thexwebModule`. Its default definition is `\arabic{xwebModule}`.

With the `noformats` option, you can suppress the printing of `@f` specifiers. (In cweb, you can also use `@s` instead of `@f`, that way you can control which format specifiers get printed and which don't).

3 Problems and Restrictions

Restrictions:

- Please be aware that the vertical bar ('`|`') is used by `WEB` to delimit small program code pieces in the documentation parts, and is therefore processed by `weave`. *You cannot use it for LATEX anymore.*

In particular, you cannot specify rules for the `tabular` or the `array` environment. Since you probably want to do so: You have two choices left:

1. Make sure you have the `array` package (by Frank Mittelbach and David Carlisle) installed. Then you may use the package `cwebarray`, it defines '`I`' (that's an uppercase i) as a specifier for rules. I.e., instead of

   ```
   \begin{tabular}{l|l}
   ```

   you have to write

   ```
   \begin{tabular}{lIl}
   ```

2. Use '`^^7c`' instead of '`|`'. I.e., instead of `\begin{tabular}{l|l}` you may write `\begin{tabular}{l^^7cl}`.

These two choices are compatible, you may use both in one document. Needless to say, I consider the first alternative the better one.

- One cannot use restricted program mode in moving arguments. Most notably, this is annoying in the titles of starred sections and in `\caption` tags.
- Neither a refinement name nor an index entry made by `@^` may consist of a *single* dot-accented term. I.e., you must not write '`@<\.O@>`', '`@^\.o@>`', or even '`@^\.{foolish}@>`'. Of course you may write '`@^\.o accent@>`' or '`@< Handle accent \.o @>`'.
- If weave is run with the `-x` option, a (harmless) error message occurs about a file that "ended while scanning use of `\end`".

3.1 Reserved Control Sequences

The following tags are reserved and must neither be used nor redefined:

```
\ATL
\B
\M
\N
\PB
\Y
```

`\9` is already explained in the `CWEB` user manual: It's a special control sequence used for the index entries tagged with '`@:`'. Its default definition is setup in such a way that you can cheat `weave` concerning the sort order of this entry: If you enter '`@:sort}{print@>`' you will get an index entry "print" next to the place where the index entry "sort" would be. But you're allowed to change this default definition.

The names of all other control sequences defined by this package—besides the common LaTeX control sequences—start with `xweb`. Please don't define new control sequences starting with this prefix. (The control sequences that don't have a '`@`' or '`_`' in them may be redefined to change the appearence of the `WEB` document, check the implementation's documentation for their meaning.)

3.2 Problems

Since this is still a test version of the package, there are some known bugs and problems.

Known Bugs:

- The presentation of `@l` redefinitions is not proper. But it wasn't in the plain version, either.

Problems:

- One cannot use an other basic font size than 10 pt. A few symbol definitions and layout parameters depend on this.
- C++ comments in `CWEB` (i.e., from `//` to the end of the line) are typeset as C comments. This is especially bad if they are used for a whole block of comment lines, as it is quite common. Please put such comment blocks in the documentation part.

4 TEXnical Data

`WEB` programmers who used the plain version before should note that the macros from `cwebmac.tex` and `webkernel.tex` are not available anymore. E.g., you cannot use `\.` to typeset typewriter material; use either `\texttt` or `\verb`, as it fits the situation. On the other hand, now you're able, for example, to use `\.` for the dot accent, `\\` will be the newline again (as usual in LATEX), you can define `\C++` for the C++ logo, etc.

Another detail for ex-plainies: The "List of Programs" that replaces the table of contents is produced by the `\listofprograms` tag (during the second LATEX run), not automatically. But this is the standard LATEX way of handling such things.

There are now two kinds of indexes, both optional: Those that are created by `weave`, which come at the end of each `WEB` program, and the index of the mother document, that is, the one that is created with `\index` commands and the `theindex` environment (possibly with the help of *MakeIndex*[5]). There are no problems in using these together, but it's for you to decide in which of them the entries must go that you specify yourself. You may, with a few definitions, get the indexes of the `WEB` files into the index of the mother document.

The modules are put in a new environment, called `xwebModule`. They are not made into `\sections` and such, and they are numbered in a single layer, in exactly the same way as in the plain version, so that the cross-reference information output by `weave` will always be right. (It refers literally to the section numbers.)

You can change this, e.g., you can let `xwebModules` be `\sections`, but you'll have to change the way cross-references are printed too. For more information, consult the documentation[6].

The `webfiles` package is documented with the `doc` package, and uses `docstrip` to create its `.sty` files. See the *LATEX Companion* or the `README` file that goes together with the `webfiles` package for instructions on how to use these to change the sources or produce typeset documentation.

This package reserves the namespace `xweb`.

4.1 Files

`webfiles.sty` is the main package file. It suffices to handle `CWEB` files.

`cwebmac.tex` is the macro package that is input on the first line of the `.tex` file that `cweave` creates. It contains macros that enable TEX to handle `CWEB`, but it can see if it's being input by LATEX (instead of TEX), and, if so, it won't define anything.

x`web.tex` is the file that is input on the first line of output of *x*`weave`, where *x*`weave` is a Spidery `weave`. It inputs `webkernel.tex`, and defines additional macros that are specific for *x*`web`. The person that made *x*`web` should make sure that these macros work with LATEX too.

`webkernel.tex` is input by *x*`web.tex`; it is the Spidery analogon of `cwebmac.tex`, but it contains no language-dependent macros: These are concentrated in the *x*`web.tex` files. If `webkernel.tex` sees that it is being input by LATEX, it inputs `swebbind.sty`, and doesn't define anything, else it defines the macros that enable TEX to handle a Spidery `WEB`.

`swebbind.sty` contains redefinitions of some macros in `webfiles.sty` that make them able to handle Spidery `WEB`s.

`webfiles.sty` and `swebbind.sty` are supplied with the `webfiles` package. So is `webkernel.tex`; this file replaces the file of the same name that goes together with Spider. The *x*`web.tex` files are supplied with their respective (Spidery) `WEB`s. `cwebmac.tex` is supplied with `CWEB`.

4.2 Bug reports

If you have any comments on the `webfiles` package that may be of interest to others, please report them to the author:

Mark Potse
Medical Physics department,
Academic Medical Center
Meibergdreef 15,
1105 AZ Amsterdam, The Netherlands.
email: `M.Potse@amc.uva.nl`

You may be able to fix bugs or customize the package after consulting the documentation that goes together with this manual.

5. The documentation driver.

The `webfiles` package is documented with `docstrip`. This code will generate the documentation. Since it is the first piece of code in the file, the documentation can be obtained by simply processing this file with LaTeX 2ε.

```
⟨*driver⟩
\documentclass[twoside]{ltxdoc}
\usepackage{xwebdoc,array,cwebarray}
\pagestyle{headings}
\renewcommand\MakePrivateLetters{\makeatletter\catcode`\_=11\relax}
\IndexPrologue{\section*{Index}%
  \markboth{Index}{Index}%
    The italic numbers denote the code lines where the corresponding
    entry is described, underlined numbers point to the definition,
    all others indicate the places where it is used.}
\EnableCrossrefs
\CodelineIndex % \RecordChanges
% \OnlyDescription
⟨/driver⟩
```

5.1 The RCS version number is extracted from the keyword string and joined with the manually-set major version number. The complete version number is also written in “`version.tex`” for use in the Makefile.

```
⟨*driver | main⟩

%%
%% $Id: webfiles.dtx,v 1.39 1999/05/26 12:40:06 potse Exp $
%%

\begingroup
  \catcode`\$=9  % ignore $
  \gdef\xwebMajorNr{0.5.}
  \def\setversion#1:#2.#3:{\xdef\xwebVersion{\xwebMajorNr #3}}
  \setversion $Revision: 1.39 $:
  \def\setdate#1: #2/#3/#4#5#6:#7:#8;{\gdef\xwebDate{#2/#3/#4#5}}
  \setdate $Date: 1999/05/26 12:40:06 $;
  \newwrite\vf \immediate\openout\vf=version.tex
  \immediate\write\vf{\xwebVersion}\immediate\closeout\vf
\endgroup
⟨/driver | main⟩
```

5.2 After that, the document can be input by the driver.

```
⟨*driver⟩
\begin{document}
   \DocInput{webfiles.dtx}
   \PrintIndex
   % \PrintChanges
\end{document}
⟨/driver⟩
```

6 Implementation.

The implementation is still somewhat messy.

A good explanation of the vocabulary used here is given by Schrod in the implementation of the `cweb` style:

> Before we start with an overview of the implementation I want to explain the `CWEB` vocabulary I use while I guide you through this document. The commonly used terms sometimes denote two entities, but for the purpose of this style we need exact terms. I've tried to stick to a "canonical" computer science terminology.
>
> I distinguish two different structures in a `CWEB` file: The *document structure* and the *program structure*.
>
> A `CWEB` document consists of a series of *sections*. Within this series some sections are especially emphasized, we call them the *main sections*. (They are also called *starred sections*, since their corresponding `CWEB` tag is `@*`.) These main sections have a title, ordinary sections are untitled. A table of contents may therefore list only the main sections. Note that there is no hierarchy in the sections, they are all on the same level, i.e., they are numbered subsequently.
>
> Each section consists of three parts: (1) the *documentation part*, (2) the *definition part*, and (3) the *program part*. Each of these parts can be empty. The documentation part is mostly text with LaTeX tags. In this text material from *restricted program mode* can appear. The definition part consists of a series of either *macro* or *format definitions*. The program part is one piece of a refinement, identified by a name (see below).
>
> A `CWEB` program consists of a tree of *refinements*. A refinement is a list of program parts with the same name, ordered in appearence. The root of the tree is the refinement with the special name `@c`. The program text is defined by the DFS (i.e., infix-order) traversal of the tree.

6.1 Before we start we declare some names for category codes. By declaring the underscore '(`_`)' as letter we can use it in our macros. As this is a LaTeX package the at sign is a letter anyhow; so we can use the "private" plain and LaTeX control sequences; and with the underscore we can make our own control sequences (*cseqs* for short) more readable. Since we have to restore this category code at the end of this macro file, we save its former value in the control sequence `\xwebCatUsCode`. This method is better than to use a group, not all cseqs must be defined global this way.

```
⟨*main | spider⟩
\catcode`\@=11
\chardef\xwebCatUsCode=\catcode`\_       % top level macro file!
\catcode`\_=11 % Catcode letter
\chardef\xwebCatEscape=0
\chardef\xwebCatOpen=1
\chardef\xwebCatClose=2
\chardef\xwebCatIgnore=9
\chardef\xwebCatLetter=11
\chardef\xwebCatOther=12
\chardef\xwebCatActive=13
⟨/main | spider⟩
```

6.2 Let's identify this package against the user and in the Log file.

```
⟨*main⟩
\ProvidesPackage{webfiles}
\begingroup
    \typeout{LaTeX package `webfiles', version \xwebVersion, \xwebDate}
\endgroup
⟨/main⟩
```

6.3 The very first (alpha) version of the `cweb` style was a style option. The version on which the `webfiles` style is based was a full style. `webfiles.sty` itself is an option again, and since the author has LaTeX 2_ε, it has become a *package*. Therefore we test if a documentclass was chosen (by testing if `\section` is defined).

```
⟨*main⟩
\ifx \section\undefined
    \PackageError{webfiles}{‘webfiles’ is a package, not a class}{%
        webfiles is not a document class, but only a package.
        Please adapt your documentclass tag appropriately.
        I.e., write \MessageBreak
        \protect\usepackage{webfiles} instead of
        \protect\documentclass{webfiles}.}
\fi
⟨/main⟩
```

7 Options.

The package options may be used in the optional argument of the `\usepackage` command, or in the optional argument of the `\webfile` command. In the second case, they only apply to a single web. We keep track of the choices with logical variables and a count register, each having a global and a local variant.

```
⟨*main⟩
\newif\ifxweb_GlobalIndex \newif\ifxwebIndex
\newif\ifxweb_GlobalRef    \newif\ifxwebRef
\newif\ifxweb_GlobalRagged \newif\ifxwebRagged
\newif\ifxweb_GlobalOC
\newif\ifxweb_GlobalHideFormats \newif\ifxwebHideFormats
\newif\ifxweb_GlobalCon    \newif\ifxwebCon
\DeclareOption{index}{\xweb_GlobalIndextrue}
\DeclareOption{noindex}{\xweb_GlobalIndexfalse}
\DeclareOption{reflist}{\xweb_GlobalReftrue}
\DeclareOption{noreflist}{\xweb_GlobalReffalse}
\DeclareOption{raggedbottom}{\xweb_GlobalRaggedtrue}
\DeclareOption{flushbottom}{\xweb_GlobalRaggedfalse}
\DeclareOption{onlychanges}{\xweb_GlobalOCtrue}
\DeclareOption{allsections}{\xweb_GlobalOCfalse}
\DeclareOption{nocon}{\xweb_GlobalConfalse}
\DeclareOption{contents}{\xweb_GlobalContrue}
\DeclareOption{hideformats}{\xweb_GlobalHideFormatstrue}
\DeclareOption{showformats}{\xweb_GlobalHideFormatsfalse}

\newcount\xweb_hypertype
\newcount\xweb_GlobalHypertype
\DeclareOption{hyperref}{\xweb_GlobalHypertype=1}
\DeclareOption{pdftex}{\xweb_GlobalHypertype=2}
\DeclareOption{nohype}{\xweb_GlobalHypertype=0}

\newcounter{xwebLopDepth}  \setcounter{xwebLopDepth}{9}
\newcounter{xwebOutlineDepth}  \setcounter{xwebOutlineDepth}{9}
\DeclareOption{nofiles}{\c@xwebLopDepth=-9}
\DeclareOption{writefiles}{\c@xwebLopDepth=9}
\ExecuteOptions{index,reflist,raggedbottom,allsections,%
                contents,showformats,nohype,writefiles}
⟨/main⟩
```

8 The interface between `cweave` and TEX.

Here we present all tags output by `cweave`[4] in an ordered fashion. First we look at those tags which are part of the 'protected interface,' ie, they are visible to a CWEB user, but he must not use them. Then we consider the private tags, some are used in the documentation part, others are needed to typeset program code, and there are a few tags for typesetting special characters in strings.

8.1 Some tags output by `cweave` are part of the protected interface even though they are not prefixed by `cweb`. We'll present them in the order they'll arrive in the document instance.

The following table specifies in the second column if this tag takes arguments. If the entry is non-empty, it's either a number listing just how many arguments are expected; then usual argument passing is used. Or it displays the context required.

`\ATL`	`#1␣#2␣`	CWEB operator: `@l`
		(how to output non-ASCII chars in `ctangle`)
		Arg. 1: hex code of mapped character
		Arg. 2: string output by `ctangle`
`\M`	1	CWEB structure tag: start of a section
		Arg. 1: section number
`\N`	`#1#2#3.`	CWEB structure tag: start of a section group
		Arg. 1: group depth, $0 \le$ `#1`
		Arg. 2: section number
		Arg. 3: section group name
`\PB`	1	restricted program mode material
		Arg. 1: program code
`\Y`		between major pieces of a program part
`\B`		start program material
`\fi`		CWEB structure tag: end of a section
`\9`	1	index entry, user defined layout
		Arg. 1: text of index entry

`\ATL` does only appear in front of the very first section. The section number is an explicit TEX number which might be followed by a 'changed flag' (see section 8.3). Note the usage of `\fi`, ie, each section must open an according `\if`. `\PB` might appear within its own argument (created by restricted program mode material in a refinement name within restricted program mode).

`\9` deserves a further explanation: It is expected, though not defined, that it expands to an empty token list. The parameter will be a sort key, actually; the real key to be typeset will appear afterwards. So a CWEB user might index TEX as '`@:TeX}{\TeX@>`', the index will feature it in the "T" section, not in the "\" section.

With the exception of `\PB` the tags above will not appear in math mode.

8.2 The following tags might appear to be public ones, but they are, in fact, never used. That's because they will be placed after the `\fi` which terminates the last section. The tags are given with their respective meaning in the Plain version:

`\ch`	`#1.`	Note which sections are changed
		Arg. 1: list of section numbers, like in `\U` and such.
`\inx`		Create index
`\fin`		Create the table of refinement names
`\con`		Create the table of contents
`\vfill\end`		output if index and all lists were suppressed

[4] This information refers to CWEB 3.0.

8.3 Some tags appear only in special circumstances and may therefore be considered as private tags. The largest part of them concern the tagging of program code, we'll have a look at them later. First we present the tags used in other areas.

Lists of section numbers occurs on several places: At the section start (where the list has actually only one element), within refinement names, in the identifier index, and for cross reference purposes. Cross references can be made at the end of a section, and in the refinement name list at the very end. Everywhere where a section number can occur it can be followed by a tag which shows that this section was changed by the changefile.

`\*`	tag after section number: this section is changed.

Within the identifier index we have also special tags. The identifiers are tagged like in the program mode, ie, with `\\` and `\&`. Remember that `\9`, listed above, appears also in the index.

`\I`	`#1,␣`	start of an index entry *Arg. 1*: index entry
`\[`	`#1]`	underlined section number in index *Arg. 1*: section number
`\.`	1	`@.` index entry *Arg. 1*: index entry

In the list of refinement names the entries are marked similar to the index entries. But note that `\I` has no arg here.

`\I`	start of a new refinement name

8.4 OK, now we can have a look at the large amount of tags used for tagging program code. First, we have those which represent directly C or C++ tokens.

`\?`	C operator: conditional expression
`\AND`	C operator: logical and
`\CM`	C operator: binary complement
`\DC`	C++ operator: scope resolution
`\E`	C operator: equivalence and equivalence sign after refinement name on it's definition
`\G`	C operator: greater or equal
`\GG`	C operator: shift right
`\I`	C operator: not equal
`\K`	C operator: assignment
`\LL`	C operator: shift left
`\MG`	C operator: pointer to struct component
`\MGA`	C++ operator: pointer to pointer to member
`\MM`	C operator: decrement
`\MOD`	C operator: modulo (actually, remainder)
`\NULL`	'quoted' identifier
`\OR`	C operator: binary or
`\PA`	C++ operator: pointer to member
`\PP`	C operator: increment
`\R`	C operator: logical negation
`\this`	'quoted' identifier
`\TeX`	'customized' identifier
`\V`	C operator: logical or
`\W`	C operator: logical and
`\XOR`	C operator: binary exclusive or
`\Z`	C operator: less or equal

Other tokens have variable parts, passed as arguments.

`\.`	1	C string *Arg. 1*: string
`\)`		discretionary break between string parts
`\&`	1	reserved identifier *Arg. 1*: identifier
`\\`	1	"normal" identifier with more than one chars *Arg. 1*: identifier
`|`	1	"normal" identifier with one char *Arg. 1*: character
`\C`	1	C comment *Arg. 1*: comment text
`\MRL`	1	C operator: combined binary operators *Arg. 1*: operators, `\K` must print as '='
`\SHC`	1	C++ comment *Arg. 1*: comment text
`\T`	1	numeric constants *Arg. 1*: constant
`\X`	`#1: #2\X`	refinement name *Arg. 1*: section number *Arg. 2*: refinement name

The refinement name (second argument of `\X`) may be a file name tagged by '`\.`'. Then the name must be set like a C string.

CWEB itself introduces its own operators:

`\ATH`		`@h` (place the preprocessor definitions here)
`\D`		`@d` (define macro)
`\F`		`@f` (format identifier like another one)
`\J`		`@&` (join)
`\vb`	1	`@=` (pass arg verbatim by `ctangle`) *Arg. 1*: string

And we have some tags used for cross referencing. Section cross-referencing (actually, for program parts) is started with some tag, then follows a list of section numbers.

`\A`	one add-on definition ("See also section")
`\As`	more than one add-on definition ("See also sections")
`\ET`	separator between the last two numbers if there are only two
`\ETs`	separator between the last two numbers if there are more than two
`\Q`	the section where it is cited ("This code is cited in section")
`\Qs`	more than one section where it is cited ("This code is cited in section")
`\U`	used in one section ("This code is used in section")
`\Us`	used in more than one section ("This code is used in sections")

Program code must be indented according to its structure. Each "larger" statement is typeset as one paragraph, usually only one line long. The basic indentation of the next statement may be incremented and decremented in given units. If a statement has to be broken in more than one line nevertheless, a hanging indentation is added to the basic indentation for the subsequent lines. One can mark an 'optional' statement start, ie, some kind of optional paragraph start. If a line break must be inserted it should be inserted here and no hanging indentation should be added to the basic indentation.

`\1`		future stmts indented one more unit
`\2`		future stmts indented one less unit
`\3`	1	optional line break within a statement *Arg. 1*: digit, penalty for line break
`\4`		backspace one indentation unit
`\5`		optional line break or small space (between run-in statements)
`\6`		line break
`\7`		line break and additional vertical space
`\8`		line is a preprocessor directive (must be issued at start of line)

8.5 Some args are designated as strings above. Within these args the following tags are used to represent special characters:

`\␣`	space
`\&`	ampersand
`\\`	backslash
`\^`	hat
`\_`	underscore
`\{`	left brace
`\}`	right brace
`\~`	tilde

8.6 Last, we have tags which are used in their "standard" LATEX meaning. We don't have much work with them:

`\#`	hash mark in program or in string
`\$`	dollar
`\%`	percent
`\{`	block start (is set in math mode)
`\}`	block end (is set in math mode)
`\,`	thin space (like the LATEX default)
`\langle`	left delimiter in include directives, or in templates
`\le`	C operator: less or equal
`\ldots`	Function protoypes: Variable number of arguments
`\rangle`	right delimiter in include directives, or in templates

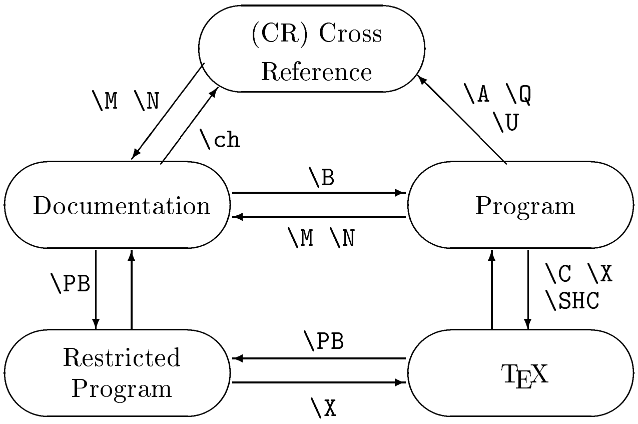

Figure 2: The processing state's FSA. The automaton starts and ends in the "documentation" state.

9 Processing states.

We have to typeset five different categories of material: Documentation, program pieces—embedded within the documentation and as large chunks, TEX material within program pieces (i.e., comments and refinement names), and cross reference information. Since we need a complete other environment for the program pieces than for the rest we design "states" where we switch to appropriately.

1. A section starts in the *documentation state.*
2. `\B` switches to *program state.* This can happen in documentation and program state.
3. While we process the argument of `\PB` we're in *restricted program state*; `\PB` may appear in documentation and in TEX state. Since TEX state can be switched on within restricted program state, `\PB` can appear within the argument of itself.
4. In the arguments of `\C`, `\SHC`, and `\X` we switch to *TEX state.* All these cseqs appear only in (restricted) program state, their official names are actually different. I.e., only in (restricted) program state these cseqs are bound to the meaning described here.
5. Cross reference information are attached to most sections with refinements. This information is processed in *CR state.* After CR state material comes always the next section or the document end, i.e., material in documentation state.

This FSA is illustrated by the diagram in figure 2.

`\xweb_documentation` will switch to documentation state, `\xweb_program` to program state, `\xweb_Rprogram` to restricted program state, `\xweb_tex` to TEX state, and `\xweb_CR` to CR state. If we're already in a state, the switch to this state shall be a permissible null operation.

9.1 The basic difference between these states can be named with two parameters: (1) The cseq bindings in effect and (2) the layout parameters used for paragraph makeup.

In (restricted) program state and in CR state the text is output under the control of `cweave`, and tagged by `cweave`. The used tags are from a global namespace and should only be in effect during these states. We call this tag set the *CWEAVE bindings.* In the other two states the tags are largely defined by the user, the tag set is called the *user bindings.* The switch to another binding is always done locally, i.e., if we switch from documentation to restricted program state within a group we don't have to bother about the restauration of the user binding; it will be done automatically by TEX at the end of the group. Nevertheless we must be able to switch from the `cweave` bindings back to the user bindings which were in effect when we activated the `cweave` bindings. This is needed for the TEX state which is always activated within (restricted) program state.

The parameters for *program layout* are really special ones since they need to support the indentation which shows the program structure. These parameters are used in program and in TEX state. The *document layout* parameters established by the user are used in the other three states.

The following table shall summarize this. C denotes `cweave` bindings, U user bindings, P program layout, and D document layout. If an entry is empty, its value is not changed on entry in this state.

State	Binding	Layout
documentation	U	D
program	C	P
restricted program	C	
TeX	U	
CR	C	D

9.2 Since the user bindings and the document layout is defined initially, we don't have to do anything if it's requested. Only if we change it, i.e., within `\xweb_CweaveBindings` and `\xweb_ProgramLayout`, we redefine `\xweb_UserBindings` and `\xweb_DocLayout`. If they are eventually executed, they shall rebind themselves back to `\relax`. This way we can switch to documentation state as often as we want.

```
⟨*main⟩
\let\xweb_UserBindings=\relax
\let\xweb_DocLayout=\relax
\def\xweb_documentation{%
    \xweb_UserBindings
    \xweb_DocLayout}
\def\xweb_Rprogram{%
    \xweb_CweaveBindings}
\def\xweb_program{%
    \xweb_CweaveBindings
    \xweb_ProgramLayout}
\def\xweb_tex{%
    \xweb_UserBindings}
\def\xweb_CR{%
    \xweb_CweaveBindings
    \xweb_DocLayout}
⟨/main⟩
```

10 Saving and restoring control sequences.

We have a lot of cseqs which are defined within the namespace of this package and which will be used with other names. This usage is in a controlled environment, namely neither in documentation nor in TeX state. (I.e., the text processed is tagged by `cweave`, not by humans; therefore we have a precise specification of the cseqs we have to accept.) We cannot work with groups where a cseq is just redefined and TeX takes care for establishing the old binding again; when we switch from program state to TeX state all the bindings which were in effect before the program state got active, i.e., in the documentation state, must be in effect again. We cannot simply consider the TeX state as something parallel to program state, it must be a hierarchical relationship: In the program state values are set up which must be available after switching back from TeX to program state.

We save the current binding of a cseq in another cseq, but only if there exists a binding currently. This is done to save valuable TeX main memory. Actually, one can assume that nearly no cseq bindings must be saved at all—the used names are strange enough. The bindings of `\foo`, i.e., of the cseq with the name `foo`, is saved as the cseq with the name `xweb_s_\foo`, i.e., as `\csname xweb\_s\_\string\foo\endcsname`.

10.1 The save process is not done statically, but by the macro `\xweb_rebind` which interprets a list of tuples (*old_name*, *new_name*), terminated by the tuple (`\stop`, `\stop`). Eventually it constructs two new lists, `\xweb_ToRestore` with the cseqs which had a binding, and `\xweb_undefined` with the names which didn't have one.

The saving is actually done by `\xweb_SaveBinding`, `\xweb_rebind` is responsible for the initialization, `\xweb_DoRebind` for the effective rebinding and the tail recursion on the list.

We could pull the `\next` assignment in the `\else` branch out of the loop to get a better performance. Should measure if this is of interest.

```
⟨*main⟩
\newtoks\xweb_undefined
\newtoks\xweb_ToRestore
\def\xweb_rebind{%
    \xweb_undefined{}%
    \xweb_ToRestore{}%
    \xweb_DoRebind
    }
\def\xweb_DoRebind#1#2{%
    \ifx #1\stop
       \let\next\relax
    \else
        \xweb_SaveBinding #2%
        \let #2=#1%
        \let\next\xweb_DoRebind
    \fi
    \next
    }
⟨/main⟩
```

10.2 If the cseq to be saved is undefined, it may just be added to the "undefined list." Otherwise its binding is saved and it's added to the "to-be-restored list."

TeXnical note: The cseq-name for the saved binding must be created before the `\let` is executed.

```
⟨*main⟩
\def\xweb_SaveBinding#1{%
    \ifx #1\undefined
        \xweb_undefined \expandafter{\the\xweb_undefined #1}%
    \else
       \expandafter\let \csname xweb_s_\string#1\endcsname =#1%
```

```
        \xweb_ToRestore \expandafter{\the\xweb_ToRestore #1}%
    \fi
    }
⟨/main⟩
```

10.3 The restoration of rebound cseqs is a two-tied activity: All previously undefined cseqs must be made undefined again, and all saved cseqs must be restored. Actually, we don't need to reset the two token lists, but we do it to save space.

Both `\xweb_undefine` and `\xweb_RestoreBinding` iterate over a list of cseqs terminated by `\stop`.

And here the `\next` assignment could be prepended to the loop as well.

```
⟨*main⟩
\def\xweb_RestoreBindings{%
    \expandafter\xweb_undefine \the\xweb_undefined \stop
    \xweb_undefined{}%
    \expandafter\xweb_RestoreBinding \the\xweb_ToRestore \stop
    \xweb_ToRestore{}%
    }
\def\xweb_undefine#1{%
    \ifx #1\stop
        \let\next\relax
    \else
        \let#1\undefined
        \let\next\xweb_undefine
    \fi
    \next
    }
⟨/main⟩
```

10.4 TEXnical note: As in `\xweb_SaveBinding`, the cseq-name for the saved binding must be created before the `\let`s are executed.

Another `\next` assignment.

```
⟨*main⟩
\def\xweb_RestoreBinding#1{%
    \ifx #1\stop
        \let\next\relax
    \else
        \expandafter\let \expandafter#1\csname xweb\string#1\endcsname
        \expandafter\let \csname xweb_s_\string#1\endcsname \undefined
        \let\next\xweb_RestoreBinding
    \fi
    \next
    }
⟨/main⟩
```

11 Hyperlinks.

Because at least two implementations of hyperlinks are supported, it is most convenient to define private macros for hyperlinks and hypertargets. Their definition depends on `\xweb_hypertype` and must therefore be performed as part of the `\webfile` command, after the options are processed. (This is more efficient than testing `\xweb_hypertype` again for each link.)

The anchor name depends on the current value of the `xwebModule` counter; this value must be set before `\xweb_hyperlink` or `\xweb_hypertarget` is used.

```
⟨*main⟩
\def\xweb_anchor{xwebmod:\xwebJobname:\arabic{xwebModule}}
\def\xweb_SetupHrefs{
   \ifnum\xweb_hypertype=0
      \def\xweb_hlink##1{##1}
      \def\xweb_htarget##1{##1}
   \else\ifnum\xweb_hypertype=1
      \def\xweb_hlink##1{\hyperlink{\xweb_anchor}{##1}}
      \def\xweb_htarget##1{\hypertarget{\xweb_anchor}{##1}}
   \else\ifnum\xweb_hypertype=2
      \def\xweb_hlink##1{\xweb_pdflink{\xweb_anchor}{##1}}
      \def\xweb_htarget##1{\xweb_pdftarget{\xweb_anchor}{##1}}
   \else\PackageError{webfiles}{This can't happen.}{%
      this may be a bug in the webfiles package}
   \fi\fi\fi}
\xweb_SetupHrefs
⟨/main⟩
```

12 PDFTEX support.

PDFTEX is an implementation of TEX that has the ability to output pdf instead of dvi. It has some extra primitives, that control pdf-specific things, such as hyperlinks, colours, and 'outlines'.

12.1 The outline is a kind of table of contents, that is always present at the left side of the pdf reader window (if the user wishes so). Web documents and main sections get outline entries.

```
⟨*main⟩
\def\xwebPDFOutline#1#2{{%
      \edef\tmpnr{#1}%
      \edef\name{#2}%
      \c@xwebModule=\tmpnr\relax
      \ifnum\xweb_hypertype=2
          \edef\anchor{\xweb_anchor}
          \pdfoutline goto name{\anchor}{#1 \name}
      \fi}}
⟨/main⟩
```

12.2 PDF links are implemented using code copied from `pdfcwebmac.tex`.

```
⟨*main⟩
   \def\xweb_pdflink#1#2{%
       \pdfannotlink
          attr{/Border [0 0 0]}
          goto name{#1}
          \Red #2\Black
       \pdfendlink}
   \def\xweb_pdftarget#1#2{
       \pdfdest name{#1} fitbh
       #2}
⟨/main⟩
```

13 Sections.

We distinguish between main sections with titles which start a group of sections and normal (untitled) sections within a group. All sections are numbered sequentially by `cweave`. The section numbers are output in boldface at the very start of a section, followed by a dot and a quad. The changeflag is a star lapping to the right.

```
⟨*main⟩
\def\xwebLapStar{\rlap{*}}
⟨/main⟩
```

13.1 Important main sections are (optionally) started on a new page, normal sections have approximately two picas vertical space in front. The very first main section does not automatically start on a new page since a title may be in front of it.

The user may also add a "level" to the title of a main section. A newpage is inserted before a section if its level is smaller than the value of the counter `xwebSecNoEject` which has default value 3. And we will pay attention to this group level when we create a table of contents.

The title of a main section is set as a heading after the section number, also in bold face. The first paragraph of the following documentation has no indentation. The documentation part of normal sections is run in after the section number.

```
⟨*main⟩
\newcount\xwebGroupLevel
\newcounter{xwebSecNoEject}
        \setcounter{xwebSecNoEject}{3}
⟨/main⟩
```

13.2 The section number that is supplied by `cweave` is used to set the counter `xwebModule`. This is necessary because the section numbers can be non-sequentially, for example, if only changed sections are printed, or if an excerpt from a web is taken. By using a counter, the appearance of the number is made changeable in the normal LaTeX way through redefinition of `\thexwebModule`. The number can contain a star, to indicate that the section is changed by a changefile. In that case, `\ifon` is made `true`; else it will be `\maybe`, and by redefining `\maybe` the user can decide wether non-changed sections should be printed. The default behaviour is to print them. This weird interface is copied from the plain version, but this package adds a friendlier interface.

This is all implemented in the macro `\xweb_PrepareSection`, which is used before each section. This macro also changes to documentation state.

```
⟨*main⟩
\newcounter{xwebModule}
\newif\ifon % determines if the section is ‘on’
\def\onmaybe{\let\ifon=\maybe} % analogous to \ontrue
\let\maybe=\iftrue  % initial meaning; the user can say "\let\maybe=\iffalse"
\def\xweb_PrepareSection#1{%
   {\xdef\secstar{#1}%
    \let\*=\empty%  remove \* from the number
    \xdef\xweb_secno{#1}}
   \ifx\xweb_secno\secstar % if they are the same; i.e.:
      \onmaybe                  % if there was no star in #1
      \xdef\xweb_star{}%
   \else
      \ontrue
      \xdef\xweb_star{*}%
   \fi
    \setcounter{xwebModule}{#1}% global set!
    \xweb_documentation
}
⟨/main⟩
```

13.3 Every web section is contained in a `xwebModule` environment. This environment takes care of vertical space before the section, optional page breaks, setting the section number, and initializing the `\everypar` token register.

`\everypar` must make `\@noskipsec` false, to indicate that the first paragraph of the documentation part has been processed. We have to initialize it ourselves, because there is no `\section` to do it. `\Y` and `\B` use `@noskipsec` to decide if there must be a `\par` before the program part.

The current reference, `\@currentlabel`, is set to the web section.

If `\xwebRaggedtrue`, the user has specified that sections should preferably not be broken across pages, so every section will be finished with optional vertical space and a negative penalty. Else, just the negative penalty is inserted to indicate that this is a good place for a page break.

The module is typeset in a `\vbox` that is `\unvbox`ed at the end of the environment. This way, the output routine cannot fire while TEX is processing a program text, and no name conflicts can arise even when the weave bindings are in effect. Thanks to Hans Hagen for the idea. —No, we can't do it this way. The documentation part cannot generate floats if it's in a vbox. Only the program part should be vboxed and unvboxed, but this is more difficult. Must think about it.

The module is also the target for hyperlinks, using `\xweb_htarget`. (This macro is defined to produce only its second argument if the hyperref package is not in use.)

If it is the first module, the table of contents is set first. By setting the table of contents when the first module is found, we allow the user to change `\xwebContentsTop` etc. in the limbo section.

xwebModule

```
⟨*main⟩
\def\thexwebModule{\arabic{xwebModule}}
\newbox\xweb_ModBox
\newif\ifxweb_FirstModule \xweb_FirstModulefalse
\newenvironment{xwebModule}{%
   \ifxweb_FirstModule
      \ifxwebCon\xwebContents{\xwebJobname}\fi
      \global\xweb_FirstModulefalse
      \openout\xweb_cont=\xwebJobname.con
      \write\xweb_cont{\catcode `\noexpand\@=11\relax}%
   \fi
   \ifxwebRagged\vfil\penalty-200\vfilneg%  (TeXbook p.353)
                                 % (this is a \filbreak without a \par)
   \else\penalty-200\fi
   %\setbox\xweb_ModBox=\vbox\bgroup
   \edef\@currentlabel{\thexwebModule}
   \par\bigskip
   \everypar{%
      \if@noskipsec%
         \global\@noskipsecfalse%
         \everypar{\penalty-100}%
      \fi}
   \@noskipsectrue
   \mbox{}\hspace{-\parindent}%
   \xwebModuleHook
   \xweb_htarget{\textbf{\thexwebModule.}\xweb_star}% typeset module number
   \hskip 1em plus0.1em minus 0.1em%
   }{%\egroup\unvbox\xweb_ModBox
   }
\def\xwebModuleHook{}
⟨/main⟩
```

13.4 The main sections are started with the macro `\N`. The parameters are:

- `#1`: a number denoting the "group level" of the main section, titles of main sections with lower numbers should be presented more prominently in the table of contents.
- `#2`: the section number.

#3: the title of the section group. This last parameter must be terminated by a dot.

The group level must be passed to the macros that format the table of contents. Since there is no easy place predetermined by LaTeX for such a wish we use a trick: The entry text for the table of contents first sets the count register `\xwebGroupLevel` with the respective level—this register may be tested after a pro-forma evaluation of the entry.

When writing to the `.con` file, we have to use `\@unexpandable@protect`; else there will be trouble with insertion of `\xweb_ModTitle` in the contents file.

We tell the user that we have reached the next starred section. The message is output after the section header is set, it shall be on the correct page. Since the token list to be output is expanded at shipout time, we must take care for the immediate expansion ourselves.

A newpage is inserted if the level of the section is smaller than the `xwebSecNoEject` counter and it is not the first section (if the user wants the web document to begin on a new page, he can insert a newpage himself).

```
⟨*main⟩
\def\xwebMainSecSkip{\clearpage}
\def\N#1#2#3.{%
    \ifon\end{xwebModule}\fi
    \global\xwebGroupLevel #1                   % <-- space!!
    { \let\*=\empty%
      \xdef\xweb_secno{#2}% get the section number
    }%
    \ifnum\xweb_secno>1
       \ifnum \xwebGroupLevel<\c@xwebSecNoEject
          \xwebMainSecSkip
       \fi
    \fi
    \message \expandafter{*\xweb_secno}%
    \xweb_PrepareSection{#2}
    \ifon
       \def\xweb_ModTitle{#3}%
       \begin{xwebModule}{\textbf{#3.}}% title
       \hskip 1em plus.1em minus.1em%
       \ifnum\c@xwebLopDepth > #1%
          \addcontentsline{lop}{starred}{%
          \protect\global\xwebGroupLevel #1 %  space after #1 !
              {\thexwebModule.}~{#3}}%
          {   \let\protect\@unexpandable@protect
              \edef\next{\write\xweb_cont{%
                \ZZ{#3}{#1}{\xweb_secno}{\thepage}}}%
              \next
          }%  write "\ZZ{title}{depth}{sec}{page}" to .con file
       \fi
       \ifnum\c@xwebOutlineDepth > #1%
          \xwebPDFOutline{\xweb_secno}{#3}% args: nr, title
       \fi
}
⟨/main⟩
```

13.5 Normal sections are started with `\M`.

```
⟨*main⟩
\def\M#1{%
    \ifon\end{xwebModule}\fi
    \xweb_PrepareSection{#1}%
    \ifon\begin{xwebModule}%
      \ifnum\c@xwebLopDepth>9%
         \addcontentsline{lop}{xwebsection}{%
```

```
          \protect\global\xwebGroupLevel 4 \thexwebModule.}%
      {   \let\protect\@unexpandable@protect
          \edef\next{\write\xweb_cont{%
            \ZZ{}{4}{#1}{\thepage}}}%
          \next
      }% write "\ZZ{title}{depth}{sec}{page}" to .con file
    \fi
  }
⟨/main⟩
```

14 The List of Programs.

The "List of Programs" is like the "List of Figures" or the "List of Tables", which are defined in the document class. The following definition is copied, with the necessary changes, from `article.sty`, 16-Mar-88. It defines `\listofprograms` which is used like `\listoffigures`.

```
⟨*main⟩
\def\xwebTocName{Contents of \xwebJobname}
\def\xwebLopName{List of Programs}
\ifx\listofprograms\undefined
  \ifx \chapter\undefined
      \def\listofprograms{\section*{\xwebLopName\@mkboth
      {\uppercase{\xwebLopName}}{\uppercase{\xwebLopName}}}\@starttoc{lop}}
  \else
      \newcommand{\listofprograms}{\chapter*{\xwebLopName\@mkboth
      {\uppercase{\xwebLopName}}{\uppercase{\xwebLopName}}}\@starttoc{lop}}
   \fi
\fi
⟨/main⟩
```

14.1 Entries for the `.lop` file are generated with `\addcontentsline` by the following commands:

- `\webfile` produces a `program` entry
- `\N` produces a `starred` entry
- `\M` produces a `xwebsection` entry

Program names are typeset with `\xweb_TTLopLine`, which sets in large typewriter type:

```
⟨*main⟩
\def\l@program#1{%
     \xwebGroupLevel\z@         % default value of group level
     \begingroup
         \let\numberline\@gobble        % width not known yet
         \setbox\z@ \hbox{#1}%
     \endgroup
     \xweb_TTLopLine{#1}%
}
⟨/main⟩
```

14.2 Normal sections are hardly ever mentioned in the list, but it will be possible. The entry consists of the number only, as they have no title.

```
⟨*main⟩
\def\l@xwebSection#1{
     \xwebGroupLevel\z@ % default value of group level
     \begingroup
         \let\numberline\@gobble        % width not known yet
         \setbox\z@ \hbox{#1}%
     \endgroup
     \xweb_NormalTocLine{#1}%
 }
⟨/main⟩
```

14.3 Titles of main sections might have an associated group level that determines how they are featured in the table. Titles on group level 0 are typeset boldface, other titles in roman. They are indented proportionally to the group level, with a basic indentation of `\xwebLopIndent`. An entry on group level 1 is not indented, the different layout suffices as a distinction to level 0. The counter `\xwebLopIndentMaxLevel` constitutes an upper limit for a recognized group level (concerning indentation, that is).

```
⟨*main⟩
\newdimen\xwebLopIndent
    \xwebLopIndent=2em
\newcount\xwebLopIndentMaxLevel
    \xwebLopIndentMaxLevel=4   % group level <= max level
⟨/main⟩
```

14.4 The group level is specified within the entry name, i.e., in the first argument to `\l@starred`. If this argument is evaluated, `\xwebGroupLevel` might be set to the respective value. If it is not set, an explicit invocation of `\addcontentsline` is responsible for this entry. Then we assume that the group level is 0.

```
⟨*main⟩
 \def\l@starred#1{%             % page will be processed later
    \xwebGroupLevel\z@          % default value of group level
    \begingroup
        \let\numberline\@gobble         % width not known yet
        \setbox\z@ \hbox{#1}%
    \endgroup
    \ifnum \xwebGroupLevel=\z@
        \let\next\xweb_BoldTocLine
    \else
        \let\next\xweb_NormalTocLine
    \fi
    \next{#1}}
⟨/main⟩
```

A `\next` assignment is needed here, because the macro is actually called with 2 arguments; the second is used by `\@dottedtocline` and such, so a `\fi` after the expansion is not allowed.

14.5 A normal line has an indentation of $(l-1)\cdot \mathit{toc\_indent}$, where $l = \min(\mathit{group\_level}, \mathit{max\_level})$.

The first argument of `\@dottedtocline` is used to specify a depth of the issued entry. All entries with a depth larger than `tocdepth` are discarded! So we always specify level 1, as the level doesn't matter to LaTeX and no entry is to be discarded.

```
⟨*main⟩
\def\xweb_NormalTocLine#1{%     % page will be processed later
   \ifnum \xwebGroupLevel>\xwebLopIndentMaxLevel
       \xwebGroupLevel\xwebLopIndentMaxLevel
   \fi
   \advance\xwebGroupLevel\m@ne        % group level -= 1
   \@dottedtocline
           % \xwebGroupLevel                  % level (old)
           1                                  % level
           {\xwebGroupLevel\xwebLopIndent}%   % basic indent
           \xwebLopIndent                     % numwidth
           {#1}%                              % entry
   }
⟨/main⟩
```

14.6 A bold line is typeset like a normal line, only the entry is typeset in bold.

```
⟨*main⟩
\def\xweb_BoldTocLine#1{%
    \@dottedtocline
            1                                   % level
            {\xwebGroupLevel\xwebLopIndent}%    % basic indent
            \xwebLopIndent                      % numwidth
            {{\bf #1}}%                         % entry
    }
⟨/main⟩
```

14.7 A `xweb@tt_lop_line` is like a `cweb@bold_toc_line` in the `cweb` style: It has no dots, but the entry is typeset in large typewriter type, instead of bold roman.

```
⟨*main⟩
\def\xweb_TTLopLine#1#2{%
    \addpenalty{\@secpenalty}%
    \addvspace{2em plus\p@}%
    \begingroup
        \noindent
        \hangindent\xwebLopIndent
        \rightskip\@tocrmarg  \parfillskip -\rightskip
        \interlinepenalty\@M
        \@tempdima\xwebLopIndent        % for \numberline
        {\large\bf\tt #1}\nobreak\hfill \hbox to\@pnumwidth{\hss #2}%
        \par
    \endgroup
    \addvspace{.5em plus\p@}%
    }
⟨/main⟩
```

15 Typesetting programs.

Program pieces come in two flavours: as argument of `\PB`, or as material after `\B`. In the former case we can use a group for the switch to the restricted program state, the group end will restore the previous state again. In the latter case we use `\B` for the switch to program state, the cross reference list or the next section will go to another state. Since we have to do more for the material after `\B`, we define this cseq later.

```
⟨*main⟩
\def\PB#1{%
    \begingroup
        \xweb_Rprogram
        \leavevmode
        #1%
    \endgroup}
⟨/main⟩
```

15.1 Note, that `\Y` cannot just be `\smallskip`, as in the plain version. We must assert that the current paragraph is ended before the vertical glue is inserted, and that's not done by the LaTeX definition of `\smallskip`. In addition we add some negative penalty, here is a good place for a page break. As the penalty value we use half the section break penalty—of course a section start is an even better place for a page break…

It might be that `\Y` is the very first token in a "normal" section. Then the section number isn't set already since it is to be run-in. In that case we don't set the vertical skip but simply start the section with the program part.

TeXnical note: The flag `@noskipsec` may be used to test if we're immediately after a run-in section heading.

```
⟨*main⟩
\newcount\xwebProgPenalty
        \xwebProgPenalty=\@secpenalty
        \divide\xwebProgPenalty by 2
\def\Y{%
    \if@noskipsec
    \else
        \par
        \penalty\xwebProgPenalty
        \smallskip
    \fi}
⟨/main⟩
```

15.2 Yep, let's unfold the "official" names of the cseqs used in program state.

If we turn on the `cweave` bindings, they might be in effect already, we don't need to establish them again. We can test `\xweb_UserBindings` for this case, it will be redefined then.

```
⟨*main⟩
\def\xweb_CweaveBindings{%
    \ifx \xweb_UserBindings\relax
        \xweb_rebind
            % indentation and paragraph layout
            \xweb_IncrIndent    \1%
            \xweb_DecrIndent    \2%
            \xweb_ExprBreak     \3%
            \xweb_backup        \4%
            \xweb_OptBreak      \5%
            \xweb_break         \6%
            \xweb_BigBreak      \7%
```

```
          \xweb_noindent      \8%
          % C/C++ tokens
          \xwebRel               \?%
          \xwebAddress           \AND
          \xwebComplement        \CM
          \xwebScope             \DC
          \xwebEquiv             \E
          \xwebGe                \G
          \xwebRightShift        \GG
          \xwebNe                \I
          \xwebAssign            \K
          \xwebLeftShift         \LL
          \xwebMod               \MOD
          \xwebNull              \NULL
          \xwebNot               \R
          \xwebBinOr             \OR
          \xwebMemberRef         \PA
          \xwebThis              \this
          \xwebOr                \V
          \xwebAnd               \W
          \xwebXor               \XOR
          \xwebLE                \Z
          \xwebPointer           \MG
          \xwebPointerMemberRef  \MGA
          \xwebDecr              \MM
          \xwebIncr              \PP
          % more tokens
          \xwebId             \\%
          \xwebIdLetter       \|%
          \xwebRes            \&%
          \xwebString         \.%             %% ( ...Emacs...
          \xwebStringBreak    \)%
          \xwebNumber         \T
          \xwebCombinedOp     \MRL
          % goes to TeX state
          \xwebComment        \C
          \xwebCxxComment     \SHC
          \xwebRefName        \X
          % CWEB tokens
          \xwebMacrosHere     \ATH
          \xwebDefine         \D
          \xwebFormat         \F
          \xwebIdCat          \J
          \xwebVerbString     \vb
          % cross reference tags
          \xwebChangeFlag     \*%
          \xwebCRAlso         \A
          \xwebCRsAlso        \As
          \xwebCRCite         \Q
          \xwebCRsCite        \Qs
          \xwebCRUse          \U
          \xwebCRsUse         \Us
          \xwebCREt           \ET
          \xwebCRsEt          \ETs
          % finish the list
          \stop\stop
      \def\xweb_UserBindings{%
          \xweb_RestoreBindings
          \let\xweb_UserBindings\relax
          }%
  \fi
  }
```

```
⟨/main⟩
```

15.3 Since most of the cweave bindings are simple and tedious coding, we'll have a look at the program layout next. Between two paragraphs there must not be any skip, the skip used in document layout is saved in \xweb_SaveParskip. A few other layout parameters from the document layout must be saved as well.

```
⟨*main⟩
\newskip\xweb_SaveParskip
\newskip\xweb_SaveRightskip
\newcount\xweb_SaveSemSFCode
\newcount\xweb_SavePretolerance
\newcount\xweb_SaveHyphenpenalty
\newcount\xweb_SaveExhyphenpenalty

\def\xweb_SaveDocLayout{%
    \xweb_SaveParskip\parskip
    \xweb_SaveRightskip\rightskip
    \xweb_SaveSemSFCode\sfcode`; % changed by \xweb_ProgramLayout
    \xweb_SavePretolerance\pretolerance
    \xweb_SaveHyphenpenalty\hyphenpenalty
    \xweb_SaveExhyphenpenalty\exhyphenpenalty
    }
\def\xweb_RestoreDocLayout{%
    \parskip\xweb_SaveParskip
    \rightskip\xweb_SaveRightskip
    \sfcode`;=\xweb_SaveSemSFCode
    \pretolerance\xweb_SavePretolerance
    \hyphenpenalty\xweb_SaveHyphenpenalty
    \exhyphenpenalty\xweb_SaveExhyphenpenalty
    }
⟨/main⟩
```

15.4 When \xweb_ProgramLayout is called for the very first time we have to set up some variables. This cannot be done in advance since they depend on values that might be changed by the user in the preamble. This setup is done in \xweb_LayoutInit.

If the program layout is already active, we must not switch it on another time. This can be tested by the current binding of \xweb_DocLayout.

```
⟨*main⟩
\def\xweb_ProgramLayout{%
    \ifx \xweb_DocLayout\relax
        \xweb_LayoutInit
        \xweb_SaveDocLayout
        \def\xweb_DocLayout{%
            \xweb_RestoreDocLayout
            \let\xweb_DocLayout\relax
            }%
        % set new values
        \parskip\z@skip
        \rightskip\z@ plus 100\p@ minus 10\p@
        \sfcode`;=3000          % same stretch factor as period.
        \pretolerance\@M
        \hyphenpenalty 9999     % strings can be broken this way
        \exhyphenpenalty\@M
    \fi
    }
⟨/main⟩
```

15.5 The unit of the basic indentation is stored in `\xwebIndentUnit`. Continuation lines are indented two units further, i.e., the initial hanging indentation is 3 units. The current hanging indentation is kept in `\xweb_indent`.

We need undiscardable items which can be used as backspaces, of one and two units, respectively. This is done best by boxes. The initialization of the boxes must be postponed until the user had the chance to change `\xwebIndentUnit`, it's done in the program state initialization.

```
⟨*main⟩
\newdimen\xwebIndentUnit   \xwebIndentUnit=1em
\newdimen\xweb_indent
\newbox\xweb_bak        % backspace one unit
\newbox\xweb_bakk       % backspace two units
\def\xweb_LayoutInit{%
    \global\setbox\xweb_bak \hbox to -1\xwebIndentUnit{}%
    \global\setbox\xweb_bakk \hbox to -2\xwebIndentUnit{}%
    \global\let\xweb_LayoutInit\relax
    }
⟨/main⟩
```

15.6 Now we can formulate how to start typesetting program pieces: We switch to program state, set the initial indentation, and add the basic indentation of the first code line.

If the code line is the very first text to be typeset in this section, we don't add the basic indentation—we're already indented from the run-in section number. But we must not start our section with `\noindent`, the section number would be typeset in the left margin then. So we just emulate an empty documentation part, the section number is now correctly set. `\B` can test for this case: `@noskipsec` is still true then.

```
⟨*main⟩
\def\B{%
    \if@noskipsec \hskip 1em \fi
    \xweb_program
    \xweb_indent 3\xwebIndentUnit  \hangindent\xweb_indent
    \ifvmode
         \if@noskipsec
            \indent             % add an empty documentation part
        \else
            \noindent \kern\xwebIndentUnit
        \fi
   \fi
}
⟨/main⟩
```

15.7 If a statement is finished, a new paragraph with the basic indentation has to be started.

An optional statement break is implemented by a low penalty which will be selected if the line has to be broken. We assume that the hanging indentation of the new line is already set correctly, and have to backup two units to get the basic indentation. It might be that the line break is not chosen by TeX, we compensate the backspace for this case. This compensation is discarded at the start of a new line.

```
⟨*main⟩
\def\xweb_break{%       % forced break, between statements
    \ifmmode\else
        \endgraf        % in LaTeX it isn't sure what \par is really...
        \noindent
        \hangindent\xweb_indent  \kern\xweb_indent
        \copy\xweb_bakk % go back to basic indentation
        \ignorespaces
    \fi
    }
```

```
\def\xweb_OptBreak{%    % optional break between statements
    \hfil \penalty\m@ne
    \hfilneg  \kern .5em  \kern 2\xwebIndentUnit   % discarded on line break
    \copy\xweb_bakk
    \ignorespaces
    }
\def\xweb_BigBreak{%    % forced break and a little extra space
    \Y
    \xweb_break
    }
\def\xweb_ExprBreak#1{% % break with penalty #1 * 10
    \hfil \penalty#10
    \hfilneg            % discarded on line break
    }
⟨/main⟩
```

15.8 When we increment the indentation, we must not forget to set the hanging indentation immediately since `\xweb_OptBreak` relies on the new value. If a continuation line is needed it will also be indented one unit more, which is ok since it should be distinguishable from the next line.

Shouldn't the hanging indentation be set anew at the end of `\xweb_DecrIndent` as well?

```
⟨*main⟩
\def\xweb_IncrIndent{%
    \global\advance\xweb_indent\xwebIndentUnit
    \hangindent\xweb_indent
    }
\def\xweb_DecrIndent{\global\advance\xweb_indent -\xwebIndentUnit}
\def\xweb_backup{\copy\xweb_bak}
\def\xweb_noindent{%                            % no indentation
    \hskip -\xweb_indent \hskip 2\xwebIndentUnit
    }
⟨/main⟩
```

16 Program (C or C++) tokens.

Since the user might want to change the (somewhat unusual) way some of the operators are typeset, we supply names for them.

```
⟨*main⟩
\let\xwebAnd=\land                % logical and, &&
\let\xwebEquiv=\equiv             % equiv sign, for ==
\let\xwebGe=\ge                   % greater or equal
\let\xwebLE=\le                   % less or equal
\let\xwebNe=\ne                   % unequal, !=
\let\xwebNull=\Lambda             % NULL pointer
\let\xwebNot=\lnot                % logical not, !
\let\xwebOr=\lor                  % logical or, ||
\let\xwebXor=\oplus               % bitwise exclusive or, ^
⟨/main⟩
```

16.1 Some symbols have to be shifted around, to save computation time we put them in boxes.

The amount of the shifting depends on the used fonts. Particularly, it depends on the usage of 10 pt fonts. We use the plain TeX names for these fonts since we need *them* especially, it's not possible to substitute them without changing these macros. If the NFSS is used with LaTeX the font names are not available, we have to assert that they exist and that they are preloaded.

But there is one exception where we try to define a general look: the increment operator `++`. We might want to use it for different logos, nowadays it's en vogue to attach `++` to all kinds of product names. We choose to use the plus sign from script size (whatever this is currently), but we must be careful that it is not used as a binary operation. Instead we convert it to an ordinary symbol and set a back-kern in between.

```
⟨*main⟩
\font\teni=cmmi10
\font\sevensy=cmsy7

\newbox\xweb_pointer              % pointer to struct component, ->
\setbox\xweb_pointer=\hbox{\kern-2pt\lower3pt\hbox{\teni\char'176}\kern1pt}

\newbox\xweb_decr                 % decrement, --
\setbox\xweb_decr=\hbox{%
        \kern .5pt
        \raise 1pt \hbox{\sevensy\char0 \kern-1pt\char0}%
        \kern .5pt
        }
\newbox\xweb_incr                 % increment, ++
\setbox\xweb_incr=\hbox{%
        \kern .05em
        \raise .1em \hbox{$\scriptstyle {+}\kern -.1em{+}$}%
        \kern .05em
        }
⟨/main⟩
```

16.2 Although the following symbols are typeset like an "ordinary C++ programmer" would expect them, we provide own module names nevertheless. They can now be changed as well, i.e., orthogonality is enhanced. (And we can use `\xweb_rebind` for assigning them to their names while we switch to program state . . .).

Must fix the comment on `\xwebMod`: Is C a "modulo language" or a "remainder language." Well, IMHO $n \% i$ with $n < 0$ shouldn't be used anyhow . . .

```
⟨*main⟩
\mathchardef\xwebAddress="2026  % ‘&’, as binary op
\let\xwebAssign==               % assignment
\let\xwebBinOr=\mid             % binary or
\def\xwebComplement{{\sim}}     % ‘~’, as ordinary symbol
\def\xwebDecr{\copy\xweb_decr}  % decrement
\def\xwebIncr{\copy\xweb_incr}  % increment
\let\xwebLeftShift=\ll          % left shift, <<
\def\xwebMod{\mathbin{\hbox{\footnotesize\rm\%}}}      % modulo/remainder, %
\def\xwebMemberRef{\mathbin{.*}}        % ptr to member (on object)
\def\xwebPointer{\copy\xweb_pointer}
\def\xwebPointerMemberRef{\mathbin{\xwebPointer*}} % ptr to member (on ptr)
\def\xwebRel{\mathrel?}         % relation operator
\let\xwebRightShift=\gg         % right shift, >>
\def\xwebScope{\kern.1em{::}\kern.1em}  % scope resolution

\def\xwebThis{\xwebRes{this}}   % reserved identifier ‘this’
⟨/main⟩
```

16.3 Some tokens don't have constant names, the name is supplied as the argument.

Identifiers are typeset in italics, reserved words and type names in boldface, and strings in typewriter.

The definition of the underscore in `\xwebRes` depends on a 10 pt type. Particularly, the height is 0.6 pt and therefore 1.5 times as high as the usual height (to get a bold impression). This height must be adapted if another size is used. Is there a font dimension which can be used as the default value for a rule?

```
⟨*main⟩
\def\xwebCombinedOp#1{\mathrel{\let\K==#1}}  % eg, += operator
\def\xwebId#1{\hbox{\it#1\/\kern.05em}} % identifier, more than one char
\def\xwebIdLetter#1{\hbox{$#1$}}        % identifier, one letter
\def\xwebRes#1{%                        % reserverd words and type names
    \hbox{\bf
        \def\_{\kern.04em\vbox{\hrule width.3em height .6pt}\kern.08em}%
        #1\/\kern .05em
        }%
    }
⟨/main⟩
```

16.4 In the plain version strings are typeset in the font `cmtex10`, which is a typewriter font with extended ASCII characters. This font does not have the usual accents and can therefore not used for typesetting national characters. (Very often they are input in some 8-bit encoding, the respective character code is made active and is substituted by a cseq which expands to the correct glyph. Of course, there are better ways, but that's the reality we have to cope with.) Instead of this special font we use the standard typewriter font. This has the further advantage that `cmtex10` may not be introduced to NFSS.

Within strings certain cseqs have a special meaning; this is introduced by `\xweb_StringSetup`. The cseqs within strings are mostly accessible by their ASCII code.

Every once in a while we have discretionary breaks in a string, denoted by `\xwebStringBreak`. This break is shown with the C convention of escaped newlines.

```
⟨*main⟩
\def\xwebString#1{%
    \hbox{\tt
        \xweb_StringSetup
        #1\kern .05em
        }%
```

```
    }
\def\xweb_StringSetup{%
    \chardef\ =`\  %            % <-- two spaces !
    \chardef\&=`\&
    \chardef\\=`\\
    \chardef\^=`\^
    \chardef\_=`\_
    \chardef\{=`\{
    \chardef\}=`\}
    \chardef\~=`\~
    \def\`{`}%
    }
\def\xwebStringBreak{\discretionary{\hbox{\tt\char`\\}}{}{}}
⟨/main⟩
```

16.5 Numbers are typeset in different ways. We use the definition of the plain macros.

Should add a specification of the possible input and an explanation of the macros below. In particular, that the closing brace after `\aftergroup` is used much later is probably not grokked by most TEX programmers.

Zdeněk Wagner notes that with NFSS `\rm` in math mode is a noop. Have to check a clean work-around.

```
⟨*main⟩
\def\xweb_oct{\hbox{$^\circ$\kern-.1em\it\aftergroup\?\aftergroup}}
\def\xweb_hex{\hbox{$^{\scriptscriptstyle\#}$\tt\aftergroup}}

\def\xwebNumber#1{%             % octal, hex, or decimal constant
    \hbox{%
        $%
            \def\?{\kern.2em}%
            \def\$##1{\egroup\sb{\,\rm##1}\bgroup}% suffix to constant
            \def\_{\cdot 10^{\aftergroup}}% power of ten (via dirty trick)
            \let\~\xweb_oct \let\^\xweb_hex
            {#1}%
        $}%
    }
⟨/main⟩
```

16.6 Comments are typeset in TEX state. We add a hook, the user shall be able to change the layout (eg, he might want another font).

Currently C++ comments are typeset like C comments. This is horrible in usual circumstances, i.e., when complete blocks of text are prefixed with `//`. We should simply catenate all these text and typeset it as one paragraph, each line prefixed by `//`. But then we have to implement an `\everyline` first, and since that's not so easy we postpone it . . .

In front of a C comment there is an optional stmt break, with 2 quad in front if the line is not broken and 1.5 quad if the line is broken.

```
⟨*main⟩
\def\xwebComment#1{%
    \5%                         % 0.5em will be discarded on line break
    \hskip 1.5em
    $/\ast\,$%
    {\xweb_tex
        \xwebCommentHook
        #1%
```

```
    }%
    $\,\ast/$%
    }
\let\xwebCommentHook\relax
\let\xwebCxxComment\xwebComment
⟨/main⟩
```

17 CWEB tokens.

We distinguish three categories of CWEB tokens: (1) Those which output constant text, (2) those which have attributes to be displayed in a special way, and (3) those which start a new structure element, namely `@d` and `@f`. Let's consider them in this order.

17.1 CWEB tokens which expand in a constant string are the identifier catenation operator ('`@&`') and the macro placement directive ('`@h`').

```
⟨*main⟩
\def\xwebIdCat{\xwebString{@\&}}
\def\xwebMacrosHere{%
    \begingroup
        \def\xwebRefNumber##1{}%
        \xwebRefName :\xwebRefMacrosHere\X
    \endgroup
    }
\def\xwebRefMacrosHere{Preprocessor Definitions}
⟨/main⟩
```

17.2 Verbatim program strings, i.e., strings passed verbatim by `ctangle` ('`@=`') are typeset like normal strings, but within a box. We use 2 pt as the separating distance, this is set locally.

```
⟨*main⟩
\def\xwebVerbString#1{{\fboxsep\tw@\p@ \fbox{\xwebString{#1}}}}
⟨/main⟩
```

17.3 The refinement names are typeset in angles, this has a long tradition. We bury the typesetting of the section number (which is the first parameter) in a macro call; the user may change this to achieve special effects.

One could think about handling special values of the section numbers differently. Eg, an empty argument is not typeset at all, a 0 triggers a marginal note about a missing definition, etc.

17.4 A refinement name may be typeset both in math and horizontal mode. The name itself is typeset in horizontal mode, of course; for the angles we need math mode. Therefore we assert at the start of the macro that we're not in math mode any more. At the end we switch back to math mode if we've started in it. This conditional switch from and back to math mode is done by `\xweb_ToggleText`.

TEXnical note: If we're in math mode `\xweb_ToggleText` must be defined globally, as it will turn off math mode and the definition would be un-made then. The second invocation of `\xweb_ToggleText` would be undefined then.

A refinement may also consist of the file name the expansion of this refinement shall be written to. This file name is tagged with `\.`, it shall be typeset like a string. But the user shall be able to use the dot accent in the refinement name as well. We check if the text consists solely of the tag and its argument; in this case we substitute `\.` with `\xwebString`. Otherwise we leave it as it is. Then a refinement name may not consist of a single dot-accented expression—well, that's highly unlikely. (Nevertheless it's documented in the user's manual)

I took the implementation strategy (i.e., the math mode toggle) from the plain version. But I don't understand why it was done this way. Why isn't it just an hbox? Isn't that much simpler? Would it break something?

```
⟨*main⟩
\def\xwebRefName#1:#2\X{%
    \ifmmode  \gdef\xweb_ToggleText{\null$\null}%
    \else  \let\xweb_ToggleText\relax
    \fi
    \xweb_ToggleText
    $\langle\,${\xweb_tex \xweb_CheckDot{#2}\xwebRefNumber{#1}}$\,\rangle$%
    \xweb_ToggleText
    }
⟨/main⟩
```

17.5 The `\xwebRefNumber` macro is used in the module names in program text; It switches to footnotesize and calls `\xwebSetModuleNumber` to create the reference text.

```
⟨*main⟩
\def\xwebRefNumber#1{%
    {\reset@font \footnotesize \kern .5em%
      \xwebSetModNrList#1.}}
⟨/main⟩
```

17.6 There are two macros to typeset module numbers: `\xwebSetModuleNumber` takes one number, and `\xwebSetModNrList` takes a comma-separated list that is terminated by a period. These macros must employ `\thexwebModule` to get consistent numbers with nonstandard definitions of `\thexwebModule`. The value of the `xwebModule` counter must be set with low-level code; we cannot use `\setcounter` because it does a global set, and we want to keep the change local, just for one-shot use. (Else the number of the current module would be wrong after the use of a module name.)

In hypertext, every section number becomes a link to the corresponding section.

```
⟨*main⟩
\def\xwebSetModuleNumber#1{{\edef\tmpnr{#1}%
   \c@xwebModule=\tmpnr\relax
   \xweb_hlink{\thexwebModule}}}
⟨/main⟩
```

17.7 If we want to handle a comma-separated list of numbers, a control sequence must be inserted before each element. The following macros, essentially copied from `pdfcwebmac.tex` (plain cweb macros modified by Andreas Scherer), insert `\xweb_TagSec` before each number, filter things like `\ET`, and then expand the list.

```
⟨*main⟩
\newtoks\xweb_toksA
\newtoks\xweb_toksB
\newtoks\xweb_toksC
\newtoks\xweb_toksD
\newcount\xweb_countA
\xweb_countA=0
\def\xweb_AddTokens#1#2{%
    \edef\addtoks{\noexpand#1={\the#1#2}}\addtoks}
\def\xweb_adn#1{\xweb_AddTokens{\xweb_toksC}{#1}%
    \global\xweb_countA=1\let\next=\maketoks}
\def\xweb_poptoks#1#2|ENDTOKS|{%
     \let\first=#1\xweb_toksD={#1}\xweb_toksA={#2}}
\def\maketoks{%
    \expandafter\xweb_poptoks\the\xweb_toksA|ENDTOKS|
    \ifx\first0\xweb_adn0
```

```
    \else\ifx\first1\xweb_adn1 \else\ifx\first2\xweb_adn2
    \else\ifx\first3\xweb_adn3 \else\ifx\first4\xweb_adn4
    \else\ifx\first5\xweb_adn5 \else\ifx\first6\xweb_adn6
    \else\ifx\first7\xweb_adn7 \else\ifx\first8\xweb_adn8
    \else\ifx\first9\xweb_adn9
    \else
        \ifnum0=\xweb_countA\else\xweb_makenote\fi
        \ifx\first.\let\next=\xweb_tdone\else
            \let\next=\maketoks
            \xweb_AddTokens{\xweb_toksB}{\the\xweb_toksD}
            \ifx\first,\xweb_AddTokens{\xweb_toksB}{\space}\fi
        \fi
    \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
    \next
}
\def\xweb_makenote{\xweb_AddTokens{\xweb_toksB}%
    {\noexpand\xweb_TagSec{\the\xweb_toksC}}%
     \xweb_toksC={}\global\xweb_countA=0}
\def\xweb_tdone{\edef\st{\global\noexpand\xweb_toksA={\the\xweb_toksB}}\st}
⟨/main⟩
```

17.8 By defining `\xweb_TagSec` to `\xwebSetModuleNumber` we can now employ `\maketoks` in `\xwebSetModNrList` to typeset a list of references.

```
⟨*main⟩
\def\xweb_TagSec#1{\xwebSetModuleNumber{#1}}
\def\xwebSetModNrList#1.{%
   \setbox0=\hbox{\xweb_toksA={#1.}\xweb_toksB={}\maketoks}%
   \the\xweb_toksA}
⟨/main⟩
```

17.9 The next macro, `\xweb_CheckDot` is a hassle to implement. We must check if it's argument consists solely of a '`\.`' tag, together with the argument to this tag. Then the '`\.`'-argument shall be typeset like a string, `\.` itself must be ignored.

The check for `\.` must not evaluate the parameter. Especially `\PB` must not be evaluated within, it would lead to havoc with the redefinition of all bindings, etc. (At least that's the empirical result—it happened; although I don't know where the problems are. Anyhow, it wasn't in the specs of the rebinding process.) All other cseqs which shouldn't be used in moving arguments may cause problems here, too. Note that `\protect` is no solution here, the user often is not aware of the presence of `cweave`-generated tags. As an example, consider the refinement `@< local variables of |foo|@>` which will lead to a call with the argument '`local variables of \PB{\\{foo}}`'. `\\` must not be evaluated here.

I use an approach where I hope that it will work—but to be honest, I'm not sure. I want to define a macro `\next` which shall have an empty expansion iff `#1` was a '`\.`' tag with its argument. Empty refinement names are not allowed so we can assume that `#1` consists of at least one token. I evaluate the first token of `#1` before the `\def` is done. With an appropriate binding a `\.` will disappear, perhaps leaving an empty token list. I also take care that `\PB` is not evaluated then, by temporary rebinding it to `\relax`. Thereby I have taken care for all cseqs `cweave` might have introduced at the very front of `#1`. Other cseqs are user-tags. If they cause problems, the user might circumvent them by adding `\protect`.

```
⟨*main⟩
\def\xweb_CheckDot#1{%
    \begingroup
      \let\.\@gobble
      \let\PB\relax
      \expandafter\def \expandafter\next \expandafter{#1}%
      \ifx \next\empty
         \gdef\next##1{\xwebString}%
```

```
      \else
         \global\let\next\relax
      \fi
    \endgroup
    \next #1
    }
⟨/main⟩
```

17.10 CWEB macro definitions (done by @d) and format directives are basically program text. When they are started we are already in program state, i.e., \B has appeared in front of it. But since we have an introductionary identifier (either #define or format) to set at the very front we increase the indentation by three units.

```
⟨*main⟩
\def\xwebDefine{\xweb_macro{\#define}}
\def\xwebFormat\\#1\\#2\par{\ifxwebHideFormats\else
  \xweb_macro{format}\\{#1} \\{#2}\par\fi}
\def\xweb_macro#1{%
    \global\advance\xweb_indent \tw@\xwebIndentUnit
    \xweb_IncrIndent
    \xwebRes{#1 }%              % <-- blank!
    }
⟨/main⟩
```

18 Section cross references and changes.

At the end of sections with refinements, `cweave` may output cross reference information: In which sections additional definitions for this refinement are found, where this refinement is used, and where it is cited. This cross reference information is always introduced by a tag followed by a non-empty list of section numbers. Different tags are used for lists with only one element and for lists with more than one element, this way the introductionary text may be adapted.

Within a list of n section numbers the first $n-1$ numbers are separated by commas with one following blank. The last two numbers are separated by `\ET` if $n = 2$, and by `\ETs` if $n > 2$. If a section is changed by the changefile, a changeflag (‘`\*`’) is appended to its number. The list is eventually terminated by a full stop.

18.1 We separate the cross reference information by a smallskip from the refinement or from a previous cross reference information. The information itself is typeset in a smaller font, as it is auxilliary, inserted stuff. The number list has a hanging indentation of `\xwebNumberListHangindent`. But beware: This isn't a dimension register, it's a macro. This way one can use ems as dimensions.

We must assert that we're in CR state while we set a cross reference.

The first parameter of a cross reference information unit is the introductionary text, the second is the number list. The parameters must be evaluated in a group—local parameter changes therein must not influence the environment. The list must be passed to `\xwebSetModNrList` to typeset (and hyperreference) them correctly.

```
⟨*main⟩
\def\xwebNumberListHangindent{2em}
\def\xwebCrossRef#1#2.{%
    \par\smallskip% (= \Y)
    \xweb_CR%
    \begingroup%
        \reset@font \footnotesize%
        \noindent \hangindent\xwebNumberListHangindent%
        #1~\xwebSetModNrList #2..\par%
    \endgroup%
    }%
⟨/main⟩
```

18.2 Well, let's define all introducing tags, which in fact start the cross reference information unit. The number list is gathered by `\xwebCrossRef`.

```
⟨*main⟩
\def\xwebCRAlso{\xwebCrossRef{See also section}}
\def\xwebCRsAlso{\xwebCrossRef{See also sections}}

\def\xwebCRCite{\xwebCrossRef{This code is cited in section}}
\def\xwebCRsCite{\xwebCrossRef{This code is cited in sections}}

\def\xwebCRUse{\xwebCrossRef{This code is used in section}}
\def\xwebCRsUse{\xwebCrossRef{This code is used in sections}}
⟨/main⟩
```

18.3 The changeflag is set as a star.

```
⟨*main⟩
\let\xwebChangeFlag=*
\def\xwebCREt{ and~}
\def\xwebCRsEt{, and~}
⟨/main⟩
```

19 Including the web file.

Because the web document is included by the `\webfile` command, we have an easy way to handle the index and list of refinements at the end; this is done by the `\webfile` command itself.

19.1 As outlined above, we want to check if there are any entries in the identifier index or the refinement list. The former is stored in the file `\xwebJobname.idx`, the latter in the file `\xwebJobname.scn`. For each of these files exists a flag named `xweb_`*ext*`Entries` (where $ext \in \{\texttt{idx}, \texttt{scn}\}$ is the extension of the appropriate file) which tells if there are any entries of the respective category.

```
⟨*main⟩
\newif\ifxweb_idxEntries
\newif\ifxweb_scnEntries
⟨/main⟩
```

19.2 We consider a file as "without entries" if (1) the file does not exist, (2) is empty, or (3) has an empty line at the very front. (In fact, case (3) means that there should not be anything behind it—but we can't test this portably.)

`\xweb_HasEntries` tests the property, it's parametrized by the extension. As a result, it sets the respective flag.

We open the file first. Then we check if it doesn't exist or if it's empty, both conditions deliver true on `\ifeof`; in this case we pretend that there was an empty line. (The LaTeX kernel already provides a macro for an empty line, `\@defpar`.) Otherwise we read the first line. At this state, the emptiness of the first line is equivalent to the non-availability of entries, we can easily construct an appropriate macro call to set the flag.

```
⟨*main⟩
\newread\xweb_TmpFile
\def\xweb_HasEntries#1{%
    \openin\xweb_TmpFile \xwebJobname.#1\relax
    \ifeof \xweb_TmpFile
        \let\next\@defpar
        \message{No #1 file present for \xwebJobname !}
    \else
        \read\xweb_TmpFile to \next
    \fi
    \csname xweb_#1Entries%
            \ifx \next\@defpar  false%
            \else  true%
            \fi
        \endcsname
    \closein\xweb_TmpFile
    }
⟨/main⟩
```

19.3 Every web document can have its own table of contents. It is stored in a file that has the web name as basename, and extension `.con`. The control sequence `\ZZ` is used to tag the entries. `\xwebContents` typesets the table of contents. `\xwebSetModuleNumber` is used here to typeset the section number and possibly create a hyperlink.

```
⟨*main⟩
\newwrite\xweb_cont
\let\ZZ=\let % now you can \write the control sequence \ZZ
\def\xweb_contentsline#1#2#3#4{%
    \ifnum#2=0 \smallbreak\fi
    \hbox to \textwidth{\xweb_consetup{#2}#1
```

```
      \rm\leaders\hbox to .5em{.\hfil}\hfil\ %
      \xwebSetModuleNumber{#3}
      \hbox to3em{\hss#4}}}
\def\xweb_consetup#1{\ifcase#1 \bf % depth -1 (@**)
  \or % depth 0 (@*)
  \or \hskip2em % depth 1 (@*1)
  \or \hskip4em % depth 2 (@*2)
  \or \hskip6em % depth 3 (@*3)
  \or \hskip8em % depth 4 (@*4)
  \or \hskip10em % depth 5 (@*5)
  \else \hskip12em \fi} % depth 6 or more
⟨/main⟩
```

19.4 PDFTEX will add an outline entry referring to the table of contents.

```
⟨*main⟩
\def\xwebContents#1{
    \def\readcontents{\IfFileExists{#1.con}{\input #1.con}{}}
    \begingroup
        \xwebContentsTop
        \ifnum\xweb_hypertype=2
           \pdfdest name{#1:contents} fith
           \pdfoutline goto name{#1:contents}{#1}
        \fi
        \hbox to \textwidth{\hfil
            \xwebSectionName\hbox to3em{\hss \xwebPageName}}
        \let\ZZ=\xweb_contentsline
        \readcontents\relax % read the contents info
        \xwebContentsBot    % print the contents page(s) and terminate
    \endgroup
}
\def\xwebSectionName{Section}
\def\xwebPageName{Page}
⟨/main⟩
```

19.5 The `\xweb_InputWebfile` macro disables `\inx` etc., adds a List Of Programs entry, inputs the web document and finishes it with some code that was stolen from `\cweb@end_document` and `\cweb@finish`. We don't need these macros themselves as there should be no `\end{document}` in the web file.

Note that there are two 'jobnames': the `\jobname` of the main document and the `\xwebJobname` of the included web document.

To get the jobname, we strip off the optional filename extension with `\setname`, using its special parameter syntax. `##1` will be the basename of the file; `##2` will be the extension, if present. `##3` will only be set if the extension has mutltiple parts; it also makes it possible to eat up the second dot, that must be there to delimit the parameter text in case there is no extension. We assume that there are no forward slashes in the filename, so the forward slash can be used to delimit the third parameter.

The flag `\xweb_FirstModule` is made true so we can find out later if we are typesetting the first module (the table of contents is set then).

```
⟨*main⟩
\def\xwebContentsTop{\vskip.2in}
\def\xwebContentsBot{\par\bigskip}
\def\xweb_InputWebfile#1{
    \def\setname##1.##2.##3/{\gdef\xwebJobname{##1}}
    \setname #1../
    \ifnum\c@xwebLopDepth > -10       % entry to the list of programs:
       \addcontentsline{lop}{program}{\xwebJobname}
    \fi
    \def\xwebContentsTop{\centerline{\textbf{\xwebJobname}}\vskip.2in}
```

```
    \def\xwebContentsBot{\par\bigskip}
    \global\xweb_FirstModuletrue
    \begingroup
       \let\inx\relax
       \let\fin\relax
       \let\con\relax
       \let\ch\xwebCRChanged
       \input{#1}
       \ifon \end{xwebModule} \fi % end the last module (if it is on).
    \endgroup
    % process the index and list of refinements ...
    \ifxwebIndex\xweb_HasEntries{idx}%
        \ifxweb_idxEntries  \xwebIdIndex{\xwebJobname.idx}\fi\fi
    \ifxwebRef\xweb_HasEntries{scn}%
        \ifxweb_scnEntries  \xwebRefList{\xwebJobname.scn}\fi\fi
    % clean up:
    \closeout\xweb_cont
    \onfalse % don't end the last section when starting the first section of
             % the next web.
    \xweb_UserBindings % back to the main document
}
⟨/main⟩
```

19.6 `\xwebIdIndex` and `\xwebRefList` have a filename argument, so that they can also be employed by the user to create an index or refinement list for any web document in any place, or to combine the indexes of several documents by processing them with the `sort` program an including the result. This can be particularly worthwile for authors of large, multi-file software packages who want to have a global index of their global variables, but they should almost certainly hack the faked index file with a perl script or something alike to mention also the filenames, because section numbers are not unique through a set of web documents.

We add two special macros for this purpose, however, to take care of things that are necessary to handle an index or reflist outside the context of a web document.

```
⟨*main⟩
\def\xwebInputIndex#1{{
  \def\setname##1.##2\\{%
      \edef\xwebJobname{##1}
      \ifx\\##2\\%
      \edef\ext{idx}
      \else
      \edef\ext{\chop##2\\}%
      \fi}
  \def\chop##1.\\{##1}
  \expandafter\setname#1.\\
  \xwebIdIndex{\xwebJobname.\ext}
}}
\def\xwebInputReflist#1{{
  \def\setname##1.##2\\{%
      \edef\xwebJobname{##1}
      \ifx\\##2\\%
      \edef\ext{scn}
      \else
      \edef\ext{\chop##2\\}%
      \fi}
  \def\chop##1.\\{##1}
  \expandafter\setname#1.\\
  \xwebRefList{\xwebJobname.\ext}
}}
⟨/main⟩
```

19.7 The `\ifstr` macro compares its first and second arguments; if they are equal it executes the third argument. It is necessary that `\edef` is used here to expand the arguments!

```
⟨*main⟩
\def\ifstr#1#2#3{%
   \edef\xweb_tempa{#1}%
   \edef\xweb_tempb{#2}%
   \ifx\xweb_tempa\xweb_tempb #3\fi
   }
⟨/main⟩
```

19.8 The `\webfile` macro has an optional argument, that may contain a comma-separated list of options. `\webfile` expands into `\xweb_file`, with the optional argument always present, but possibly empty.

```
⟨*main⟩
\def\webfile{\@ifnextchar[%
   {\xweb_file}{\xweb_file[]}}
⟨/main⟩
```

19.9 The square-braced argument is mandatory for `\xweb_file`. The comma-separated list of options is processed using a `\@for` loop. The macro will complain if an unknown option is encountered. After that, `\xweb_InputWebfile` is called to do the real work. Default values for all options will be copied from their global equivalents (specified with the `\usepackage` command).

```
⟨*main⟩
\newif\if@opt
\def\xweb_off{\let\maybe=\iffalse}
\def\xweb_on{\let\maybe=\iftrue}
\def\xweb_file[#1]#2{
   \ifxweb_GlobalIndex\xwebIndextrue\else\xwebIndexfalse\fi
   \ifxweb_GlobalRef\xwebReftrue\else\xwebReffalse\fi
   \ifxweb_GlobalRagged\xwebRaggedtrue\else\xwebRaggedfalse\fi
   \ifxweb_GlobalOC\xweb_off\else\xweb_on\fi
   \ifxweb_GlobalCon\xwebContrue\else\xwebConfalse\fi
   \ifxweb_GlobalHideFormats\xwebHideFormatstrue\else
                              \xwebHideFormatsfalse\fi
   \xweb_hypertype=\xweb_GlobalHypertype
   \@for\xweb_opt:=#1\do{
     \typeout{webfiles option: \xweb_opt}
     \@optfalse
     \ifstr{\xweb_opt}{index}{\xwebIndextrue\@opttrue}
     \ifstr{\xweb_opt}{noindex}{\xwebIndexfalse\@opttrue}
     \ifstr{\xweb_opt}{reflist}{\xwebReftrue\@opttrue}
     \ifstr{\xweb_opt}{noreflist}{\xwebReffalse\@opttrue}
     \ifstr{\xweb_opt}{raggedbottom}{\xwebRaggedtrue\@opttrue}
     \ifstr{\xweb_opt}{flushbottom}{\xwebRaggedfalse\@opttrue}
     \ifstr{\xweb_opt}{onlychanges}{\xweb_off\@opttrue}
     \ifstr{\xweb_opt}{allsections}{\xweb_on\@opttrue}
     \ifstr{\xweb_opt}{nocon}{\xwebConfalse\@opttrue}
     \ifstr{\xweb_opt}{contents}{\xwebContrue\@opttrue}
     \ifstr{\xweb_opt}{hideformats}{\xwebHideFormatstrue\@opttrue}
     \ifstr{\xweb_opt}{showformats}{\xwebHideFormatsfalse\@opttrue}
     \ifstr{\xweb_opt}{hyperref}{\xweb_hypertype=1\@opttrue}
     \ifstr{\xweb_opt}{pdftex}{\xweb_hypertype=2\@opttrue}
     \ifstr{\xweb_opt}{nohype}{\xweb_hypertype=0\@opttrue}
     \if@opt\else\PackageError{webfiles}{%
         Unknown option for the \protect\webfile\space command
```

```
        }{Possible options are: index, noindex,
          reflist, noreflist, \MessageBreak
          raggedbottom, flushbottom, onlychanges,
          allsections,\MessageBreak
          nocon, contents, hideformats, showformats,
          hyperref, pdftex, nohype}
     \fi
   }
   \xweb_SetupHrefs
   \xweb_InputWebfile{#2}}
\let\cwebfile\webfile
⟨/main⟩
```

19.10 The list of changed sections is a cross reference list, (nearly) like all others at the end of a section. The only difference is that we do not show changeflags any more—each section number in this list carries a change flag by definition.

TEXnical note: The redefinition of `\*` is part of the second argument of `\xwebCrossRef`. It is *not* a global redefinition.

```
⟨*main⟩
\def\xwebCRChanged{%
    \xwebCrossRef{The following sections were changed by the change file:}%
    \let\*\relax
    }
⟨/main⟩
```

19.11 The identifier index is available in the file `\xwebJobname.idx`. The setup for the index is a mixture of the `theindex` environment of the `article` style and DEK's index macros. It's typeset in two columns; the user may specify an introductionary text for the index by `\xwebIndexIntro`. If there is any introductionary text we add a medium skip below.

The paragraph layout is taken from the plain version: Each index entry is a paragraph, nearly no skip between the paragraphs (just a bit to prevent underfull vboxes), no paragraph indentation, ragged right—but the `\parfillskip` set in such a way that almost empty lines are avoided. Overfull hboxes in the index doesn't make sense, so we prevent them. And we don't allow hyphenation in the index, it's an identifier index after all.

Before we read in this file, we have to bind the special cseqs used therein.

```
⟨*main⟩
\newtoks\xwebIndexIntro
        \xwebIndexIntro={}
\def\xwebIndexName{Index of \texttt{\xwebJobname}}
⟨/main⟩
```

19.12 The following text is set as the header of the index.

```
⟨*main⟩
\def\xweb_IndexTop{
    \xweb_tex
    \par\vskip 1cm plus 10mm minus 5mm\noindent
    {\bf \xwebIndexName}%\nopagebreak\par\medskip\nopagebreak
                        %% skip already inserted by xweb_twocolumn env.
    \edef\intro{\the\xwebIndexIntro}% % is a local def
    \ifx \intro\empty
    \else
       \noindent\the\xwebIndexIntro\unskip
       \par\medskip
```

```
    \fi
}
⟨/main⟩
```

19.13 If the multicol package is available, it will be used to typeset the identifier index. The \RequirePackage command must however be placed after \ProcessOptions, at the end of the package file.

If the twocolumn class option is in effect, multicols cannot be used; in particular, the title (ieee.cls) would be set in one column. It isn't necessary too, we are in two columns already.

```
⟨*main⟩
\newif\ifxweb_multicol \xweb_multicolfalse
\IfFileExists{multicol.sty}{
  \if@twocolumn\else\xweb_multicoltrue\fi
}{}
\ifxweb_multicol
  \newenvironment{xweb_twocolumn}{\begin{multicols}{2}}{\end{multicols}}
\else
  \newenvironment{xweb_twocolumn}{
      \if@twocolumn\else
        \columnseprule\z@
        \columnsep 35\p@          % value is from article.sty
      \fi
      \twocolumn%
  }{
      \onecolumn}
\fi
⟨/main⟩
```

19.14 Now \xwebIdIndex itself. PDFTEX will add an outline entry for the index.

vfil before \xweb_IndexTop is still not strong enough! A pagebreak can occur just after it.

```
⟨*main⟩
\def\xwebJobName{noname}
\def\xweb_SetupIndex{%
    \begin{xweb_twocolumn}[\xweb_IndexTop]
    \ifnum\xweb_hypertype=2
       \pdfdest name{\xwebJobname:index} fith
       \pdfoutline goto name{\xwebJobname:index}{Index}
    \fi
    % go to CR mode
    \xweb_CR
    \message{*index: }% % tell the user what we're doing
    % paragraph layout
    \begingroup
    \parskip \z@ plus .5\p@
    \parindent\z@
    \rightskip \z@ plus 2.5em
    \parfillskip \z@ plus .6\hsize
    \tolerance\@M \hyphenpenalty\@M
    % bindings
    \let\I\xwebIndexEntry
    \let\[\xwebIndexDeclared
    \let\*\xwebLapStar
}
\def\xweb_FinishIndex{
    \endgroup
```

```
    \end{xweb_twocolumn}
}
\def\xwebIdIndex#1{%
    \xweb_SetupIndex
    \@input{#1}
    \xweb_FinishIndex
}
⟨/main⟩
```

19.15 An index entry is typeset with the same hanging indentation as‘ a cross reference list.

The entry is tagged with `\.`, if it was entered by the CWEB operator ‘`@.`’. Then it shall be typeset as a string. But an indexed name may also want to use `\.` as an accent. This is the same situation we had at the refinement names (see section 17.4), where we introduced `\xweb_CheckDot` to handle this case. The same minor restriction as there holds here, a refinement name may not consist of a single dot-accented expression. (`@:` helps in this singular case.)

```
⟨*main⟩
\def\xwebIndexEntry#1,#2.{%
    \par
    \hangindent\xwebNumberListHangindent
    \leavevmode
    \xweb_CheckDot{#1}:\quad\xwebSetModNrList #2.
    }
⟨/main⟩
```

19.16 The sections where identifiers are declared are noted with underlined numbers. We also must not forget the default declaration of `\9`, the tag for the user definable index layout.

```
⟨*main⟩
\def\xwebIndexDeclared#1]{\underline{#1}}
\def\9#1{}
⟨/main⟩
```

19.17 The list of the refinement names is available in the file `\xwebJobname.scn`. The layout is taken from the plain version: ragged right, each entry is a paragraph, the different cross reference categories are separated by a quad.

We must restore `\parfillskip` and `\*` since it was changed in the index.

```
⟨*main⟩
\def\xwebReflistName{List of Refinements in \texttt{\xwebJobname}}
\def\xwebRLCite{\xwebCrossRef{Cited in section}}
\def\xwebRLsCite{\xwebCrossRef{Cited in sections}}
\def\xwebRLUse{\xwebCrossRef{Used in section}}
\def\xwebRLsUse{\xwebCrossRef{Used in sections}}
⟨/main⟩
```

```
⟨*main⟩
\def\xweb_SetupReflist{%
    \vskip 3ex plus 7ex minus 1ex
    \xweb_tex
    \noindent
    {\bf \xwebReflistName}\nopagebreak\par\bigskip\nopagebreak
    \ifnum\xweb_hypertype=2
       \pdfdest name{\xwebJobname:reflist} fith
       \pdfoutline goto name{\xwebJobname:reflist} {Refs}
    \fi
```

```
    \xweb_CR
    \message{*list of refinements: }%
    % paragraph layout: like in index, but
    \parfillskip\@flushglue
    \begingroup    % different bindings...
       \def\I{\par \hangindent\xwebNumberListHangindent}%
       \def\xwebCrossRef##1##2.{\quad {\reset@font\footnotesize
          ##1~\xwebSetModNrList ##2..}}%
       \def\Q {\xwebRLCite}%
       \def\Qs{\xwebRLsCite}%
       \def\U {\xwebRLUse}%
       \def\Us{\xwebRLsUse}%
       \let\*\xwebChangeFlag
}
\def\xwebRefList#1{%
       \xweb_SetupReflist
       \@input{#1}
    \endgroup
    }
⟨/main⟩
```

The redefinition of `\xwebCrossRef` etc. must be done locally to prevent troubles with trailing web files: Crossref info in these files would be indented!

20 Bells and whistles.

Just copied from the plain version. It doesn't work anyhow—`\char"`*xy* usually doesn't typeset the correct character. That's because this character is most probably an active character; at least that's typical for the way TeX systems are used in Europe. I have to think about the implementation.

```
⟨*main⟩
\def\ATL{\par\noindent\bgroup\catcode`\_=12 \postATL} % print @l in limbo
\def\postATL#1 #2 {\bf letter \\{\uppercase{\char"#1}}
   tangles as \tentex "#2"\egroup\par}
\def\noATL#1 #2 {}
\def\noatl{\let\ATL=\noATL} % suppress output from @l
⟨/main⟩
```

20.1 Option processing will be done at the end of the package file. We must restore our catcode and are finished.

```
⟨*main⟩
\ProcessOptions
\ifxweb_multicol
    \RequirePackage{multicol}
\fi
\catcode`\_=\xwebCatUsCode
%\endinput
⟨/main⟩
```

21 Spidery Webs.

The file `swebbind.sty` redefines some macros to let the above things work with web systems generated with Spider. This file is input by `webkernel.tex`, as provided with the webfiles distribution. As it is eventually input by the `\webfile` command, it is inside a group that encloses the entire web document, so the scope of these redefinements is automatically limited to a single web: we need no “`cwebbind.sty`” or such.

21.1 First, make the underscore usable in private control sequences, as in `webfiles.sty`:

```
⟨*spider⟩
\catcode`\_=\xwebCatLetter
\catcode`\@=\xwebCatLetter
⟨/spider⟩
```

21.2 Some miscellaneous…

```
⟨*spider⟩
\let\amp\&     % ampersand
\let\SS\S      % section sign
\let\PP\P      % paragraph sign
⟨/spider⟩
```

21.3 The main section tag will be a mixture of `webfiles` and `webkernel.tex`:
`#1` = number
`#2` = (level and) title

```
⟨*spider⟩
\def\N#1.#2.{%
   \ifon\end{xwebModule}\fi   % webfiles
   \global\xwebGroupLevel 0% default value for group level
   \xweb_headcheck#2\xweb_headcheck     % spider: get group level and title
   {\let\*=\empty%
      \xdef\xweb_secno{#1}% webfiles: get the section number
    }%
    \message \expandafter{*\xweb_secno}%
    \xweb_PrepareSection{#1}%
    \ifon\begin{xwebModule}{\bf\xweb_ModTitle.}%
               % \xweb_ModTitle is generated by \xweb_headcheck.
       %\hskip 1em plus.1em minus.1em%
       \xweb_SpiderLopEntry{\xwebGroupLevel}%
}
⟨/spider⟩
```

21.4 The entries in the list of programs and the web's own table of contents are handled by `\xweb_SpiderLopEntry`. We have to use `\@unexpandable@protect`; else there will be trouble with insertion of `\xweb_ModTitle` in the contents file.

```
⟨*spider⟩
\def\xweb_SpiderLopEntry#1{%
    \ifnum\c@xwebLopDepth > \xwebGroupLevel
       \addcontentsline{lop}{starred}{%
       \protect\global\xwebGroupLevel #1 \thexwebModule.~\xweb_ModTitle}%
       {   \let\protect\@unexpandable@protect
           \edef\next{\write\xweb_cont{%
             \ZZ{\xweb_ModTitle}{\the\xwebGroupLevel}%
```

```
                {\xweb_secno}{\noexpand\thepage}}}%
            \next
        }% write "\ZZ{title}{depth}{sec}{page}" to .con file
    \fi
    \ifnum\c@xwebOutlineDepth > \xwebGroupLevel
       \xwebPDFOutline{\xweb_secno}{\xweb_ModTitle}% args: nr, title
    \fi}
⟨/spider⟩
```

21.5 The following is from `webkernel`, with small additions for `webfiles`. This stuff is to allow inital =, 1, 2, 3, 4 in starred modules. = means "part", don't skip page. Normal starred module is `1`. 2–4 are submodules, and are indented.

`@*=`	bold name in table of contents
	causes page eject
	suppresses page eject following
`@*1,2`	first level of indentation
`@*3,4`	second level of indentation
`@*1,3`	cause page eject
`@*2,4`	don't cause page eject

```
⟨*spider⟩
\newif\ifxweb_cancel\xweb_canceltrue
\def\xweb_ifnextchar#1#2#3{\let\@tempe=#1\def\@tempa{#2}\def\@tempb{#3}%
   \xweb_ifnch}
\def\xweb_ifnch{%
    \ifx \@tempc \@tempe\let\@tempd\@tempa
    \else\let\@tempd\@tempb\fi
    \@tempd}
\def\xweb_makethechar#1{\let\@tempc=#1}

\def\xweb_headcheck#1#2\xweb_headcheck{%
    \xweb_makethechar{#1}%
    \def\theskipper{\vskip 3pt}%
                % extra skip before new starred module
    \def\xweb_ModTitle{{#2}}%
    \xweb_ifnextchar={%                                  % @*= title.
        \global\xwebGroupLevel 0% webfiles
        \ifnum \xwebGroupLevel<\c@xwebSecNoEject
           \def\theskipper{\xwebMainSecSkip}%
        \fi
        \xweb_canceltrue
    }{\xweb_ifnextchar1{%                                % @*1 title.
        \global\xwebGroupLevel 1% webfiles
        \xweb_cancelfalse
        \ifnum \xwebGroupLevel<\c@xwebSecNoEject
           \def\theskipper{\xwebMainSecSkip}%
        \fi
    }{\xweb_ifnextchar2{%                                % @*2 title.
        \global\xwebGroupLevel 2% webfiles
        \xweb_cancelfalse
    }{\xweb_ifnextchar3{%                                % @*3 title.
        \global\xwebGroupLevel 3% webfiles
        \xweb_cancelfalse
        \ifnum \xwebGroupLevel<\c@xwebSecNoEject
           \def\theskipper{\xwebMainSecSkip}%
        \fi
    }{\xweb_ifnextchar4{%                                % @*4 title.
        \global\xwebGroupLevel 4% webfiles
        \xweb_cancelfalse
```

```
    }{% else                                            % @* title.
        \global\xwebGroupLevel 0% webfiles
        \ifxweb_cancel\else
           \ifnum \xwebGroupLevel<\c@xwebSecNoEject
              \def\theskipper{\xwebMainSecSkip}%
           \fi
        \fi
        \xweb_cancelfalse
        \def\xweb_ModTitle{#1{#2}}%
    }}}}}%
    \theskipper
}
⟨/spider⟩
```

21.6 Surround text in vertical bars: (already tagged with \PB in Spider 3.0)

```
⟨*spider⟩
\def\CD#1\DC{#1}
⟨/spider⟩
```

21.7 Go into program mode: this is \P in spider and \B in cweb.

```
⟨*spider⟩
\def\P{\B}
%\let\P\B
⟨/spider⟩
```

21.8 Spidery WEAVE Bindings: these are a little different from the CWEAVE bindings.

```
⟨*spider⟩
\def\xweb_CweaveBindings{%
    \ifx \xweb_UserBindings\relax
    \xweb_rebind
        % indentation and paragraph layout
        \xweb_break            \0%          % SPIDER
        \xweb_IncrIndent       \1%
        \xweb_DecrIndent       \2%
        \xweb_ExprBreak        \3%
        \xweb_backup           \4%
        \xweb_OptBreak         \5%
        \xweb_break            \6%
        \xweb_BigBreak         \7%
        \xweb_noindent         \8%
        % C/C++ tokens
        \xwebRel               \?%
        \xwebAddress           \AND
        \xwebComplement        \CM
        \xwebScope             \DC
        \xwebEquiv             \S               % SPIDER
        \xwebGe                \G
        \xwebRightShift        \GG
        \xwebNe                \I
        \xwebAssign            \K
        \xwebLeftShift         \LL
        \xwebMod               \MOD
        \xwebNull              \NULL
        \xwebNot               \R
```

```
        \xwebBinOr              \OR
        \xwebMemberRef          \PA
        \xwebThis               \this
        \xwebOr                 \V
        \xwebAnd                \W
        \xwebXor                \XOR
        \xwebLE                 \L              % SPIDER
        \xwebPointer            \MG
        \xwebPointerMemberRef   \MGA
        \xwebDecr               \MM
        \xwebIncr               \PP
        % more tokens
        \xwebId                 \\%
        \xwebIdLetter           \|%
        \xwebRes                \&%
        \xwebString             \.%         %% ( ...Emacs...
        \xwebStringBreak        \)%
        \xwebNumber             \O              % SPIDER
        \xwebCombinedOp         \MRL
        % goes to TeX state
        \xwebComment            \C
        \xwebCxxComment         \SHC
        \xwebRefName            \X
     \xwebSpiderOutputFileName  \XF
        % CWEB tokens
        \xwebMacrosHere         \ATH
        \xwebDefine             \D
        \xwebFormat             \F
        \xwebIdCat              \J
        \xwebVerbString         \=            % SPIDER
        % cross reference tags
        \xwebChangeFlag         \*%
        \xwebCRAlso             \A
        \xwebCRsAlso            \As
        \xwebCRCite             \Q
        \xwebCRsCite            \Qs
        \xwebCRUse              \U
        \xwebCRsUse             \Us
        \xwebCREt               \ET
        \xwebCRsEt              \ETs
        % finish the list
        \stop\stop
    \def\xweb_UserBindings{%
        \xweb_RestoreBindings
        \let\xweb_UserBindings\relax
        }%
    \fi
    }
⟨/spider⟩
```

21.9 \B in spider is Begin controlled comment. But there are no controlled comments in Spider, so we can leave it out.

`\def\B{\mathopen{\.{@\commentbegin}}}`

21.10 In spider, the argument of \M is ended by a period. The meaning of the tag is the same.

```
⟨*spider⟩
\def\M#1.{%
    \ifon\end{xwebModule}\fi
```

```
    \xweb_PrepareSection{#1}%
    \ifon\begin{xwebModule}
       \ifnum\c@xwebLopDepth>9
       \addcontentsline{lop}{xwebsection}{%
          \protect\global\xwebGroupLevel 4 \thexwebModule.}%
          \edef\next{\write\xweb_cont{\ZZ{}{4}{\xweb_secno}%
         {\thepage}}}\next % \ZZ{title}{depth}{sec}{page}%
          {   \let\protect\@unexpandable@protect
      \edef\next{\write\xweb_cont{%
        \ZZ{}{4}{\xweb_secno}{\thepage}}}%
      \next
          }% write "\ZZ{title}{depth}{sec}{page}" to .con file
       \fi
    }
⟨/spider⟩
```

21.11 The index and the list of refinements are not given in `.inx` and `.scn` files, but in the `.tex` file itself. Furthermore, they are tagged a little differently:

`\inx`
⟨*index*⟩
`\fin`
⟨*reflist*⟩
`\con`

```
⟨*spider⟩
\def\inx{
   \ifxwebIndex
      \xweb_SetupIndex
      \def\:{\I}
   \else
      \def\:##1.{}  % gobble \:\\{foo} 1, 2, 3.
   \fi
}
\def\fin{
   \ifxwebIndex
      \xweb_FinishIndex
   \fi
   \ifxwebRef
      \xweb_SetupReflist
      \def\:{\I}
   \else
      \def\:{}
      \def\XF##1\XF{}
      \def\X##1\X{}
      \def\U##1.{}
   \fi
}
\def\con{
   \ifxwebRef
      \endgroup
   \fi
}
⟨/spider⟩
```

21.12 Comments in Spider are not fixed; rather, the cseqs `\commentbegin` and `\commentend` are set in `xweb.tex`.

```
⟨*spider⟩
\def\xwebComment#1{%
    \5%                 % 0.5em will be discarded on line break
    \hskip 1.5em
    \commentbegin%
    {\xweb_tex
    \xwebCommentHook
    #1%
    }%
    \commentend%
    }
\let\xwebCommentHook\relax
\let\xwebCxxComment\xwebComment
⟨/spider⟩
```

21.13 Macro definitions are not translated into preprocessor directives; Spidery webs expand these themselves. Therefore they should not look like preprocessor directives: the '`#`' must be left out.

```
⟨*spider⟩
\def\xwebDefine{\xweb_macro{define}}
%\def\xwebFormat{\4\xweb_macro{format}}
\def\xwebFormat\\#1\\#2\par{\ifxwebHideFormats\else
  \4\xweb_macro{format}\\{#1} \\{#2}\par\fi}
⟨/spider⟩
```

21.14 The sequence `\XF#1:#2\XF` tags output files, instead of `\X#1:\.{#2}\X`. (`\XF` is bound to `\xwebSpiderOutputFileName`)

```
⟨*spider⟩
\def\xwebSpiderOutputFileName#1:#2\XF{\xwebRefName{#1}:\.{#2}\X}
⟨/spider⟩
```

21.15 Restore catcodes:

```
⟨*spider⟩
\catcode`\_=\xwebCatUsCode
\catcode`\@=\xwebCatOther
⟨/spider⟩
```

References

[1] Sylvio Levy. *The* `CWEB` *System of Structured Documentation.*

[2] Norman Ramsey. *The Spidery* `WEB` *system of Structured Documentation.*

[3] Mark Potse. *The* `MWEB` *System of Structured Documentation*, 1994, 1997.

[4] Sebastian Rahtz. *Hypertext marks in LaTeX, the hyperref package*, October 1997.

[5] Leslie Lamport. *MakeIndex: An Index Processor for LaTeX*, February 1987.

[6] Mark Potse. *The* `webfiles` *package.*, December 1997.

Index

The italic numbers denote the code lines where the corresponding entry is described, underlined numbers point to the definition, all others indicate the places where it is used.