

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

¹version 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 L^AT_EX 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 L^AT_EX 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` option². You can specify an introductory text for the index with the tag `\webIndexIntro`, 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`.

²The ‘-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 (yet³).

The tables of contents are produced like all L^AT_EX's tables of contents etc., in the second L^AT_EX run. They are stored in a file with name $\langle\text{basename}\rangle.con$, where $\langle\text{basename}\rangle$ 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 L^AT_EX 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 its table of contents. As with the List of Figures, the List of Programs is generated at the second L^AT_EX 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`:

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

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

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 PDF_TE_X) 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 PDF_TE_X can be enabled. This provides the same look and feel as the `pdfcwebmac` macros, including the ‘PDF Outline’, but of course works only with PDF_TE_X.

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 L_AT_EX 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}{l|Il}
```

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 ‘`@<\.0@>`’, ‘`@^\.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 appearance 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 10pt. 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 T_EXnical 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 L^AT_EX), 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 L^AT_EX run), not automatically. But this is the standard L^AT_EX 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 *L^AT_EX 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 T_EX to handle CWEB, but it can see if it's being input by L^AT_EX (instead of T_EX), and, if so, it won't define anything.

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

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

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

`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 `xweb.tex` files are supplied with their respective (Spidery) WEBS. `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 $\text{\LaTeX}2_{\epsilon}$.

```

1 (*driver)
2 \documentclass[twoside]{ltxdoc}
3 \usepackage{xwebdoc,array,cwebarray}
4 \pagestyle{headings}
5 \renewcommand\MakePrivateLetters{\makeatletter\catcode'\_ =11\relax}
6 \IndexPrologue{\section*{Index}%
7   \markboth{Index}{Index}%
8   The italic numbers denote the code lines where the corresponding
9   entry is described, underlined numbers point to the definition,
10  all others indicate the places where it is used.}
11 \EnableCrossrefs
12 \CodelineIndex % \RecordChanges
13 % \OnlyDescription
14 </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.

```

15 <*driver | main>
16
17 %%
18 %% $Id: webfiles.dtx,v 1.39 1999/05/26 12:40:06 potse Exp $
19 %%
20
21 \begingroup
22   \catcode'\$=9 % ignore $
23   \gdef\xwebMajorNr{0.5.}
24   \def\setversion#1:#2.#3:#{\xdef\xwebVersion{\xwebMajorNr #3}}
25   \setversion $Revision: 1.39 $:
26   \def\setdate#1: #2/#3/#4#5#6:#7:#8;{\gdef\xwebDate{#2/#3/#4#5}}
27   \setdate $Date: 1999/05/26 12:40:06 $;
28   \newwrite\vf \immediate\openout\vf=version.tex
29   \immediate\write\vf{\xwebVersion}\immediate\closeout\vf
30 \endgroup
31 </driver | main>

```


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

```
32 <*driver>
33 \begin{document}
34   \DocInput{webfiles.dtx}
35   \PrintIndex
36   % \PrintChanges
37 \end{document}
38 </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 appearance. 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.

```

39 ⟨*main | spider⟩
40 \catcode'\@=11
41 \chardef\xwebCatUsCode=\catcode'\_      % top level macro file!
42 \catcode'\_ =11 % Catcode letter
43 \chardef\xwebCatEscape=0
44 \chardef\xwebCatOpen=1
45 \chardef\xwebCatClose=2
46 \chardef\xwebCatIgnore=9
47 \chardef\xwebCatLetter=11
48 \chardef\xwebCatOther=12
49 \chardef\xwebCatActive=13
50 ⟨/main | spider⟩

```

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

```

51 ⟨*main⟩
52 \ProvidesPackage{webfiles}
53 \begingroup
54   \typeout{LaTeX package 'webfiles', version \xwebVersion, \xwebDate}
55 \endgroup
56 ⟨/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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}2_{\epsilon}$, it has become a *package*. Therefore we test if a documentclass was chosen (by testing if `\section` is defined).

```
57 <*main>
58 \ifx \section\undefined
59   \PackageError{webfiles}'webfiles' is a package, not a class}{%
60     webfiles is not a document class, but only a package.
61     Please adapt your documentclass tag appropriately.
62     I.e., write \MessageBreak
63     \protect\usepackage{webfiles} instead of
64     \protect\documentclass{webfiles}.}
65 \fi
66 </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.

```

67 <*main>
68 \newif\ifxweb_GlobalIndex \newif\ifxwebIndex
69 \newif\ifxweb_GlobalRef \newif\ifxwebRef
70 \newif\ifxweb_GlobalRagged \newif\ifxwebRagged
71 \newif\ifxweb_GlobalOC
72 \newif\ifxweb_GlobalHideFormats \newif\ifxwebHideFormats
73 \newif\ifxweb_GlobalCon \newif\ifxwebCon
74 \DeclareOption{index}{\xweb_GlobalIndextrue}
75 \DeclareOption{noindex}{\xweb_GlobalIndexfalse}
76 \DeclareOption{reflist}{\xweb_GlobalReftrue}
77 \DeclareOption{noreflist}{\xweb_GlobalReffalse}
78 \DeclareOption{raggedbottom}{\xweb_GlobalRaggedtrue}
79 \DeclareOption{flushbottom}{\xweb_GlobalRaggedfalse}
80 \DeclareOption{onlychanges}{\xweb_GlobalOCtrue}
81 \DeclareOption{allsections}{\xweb_GlobalOCfalse}
82 \DeclareOption{nocon}{\xweb_GlobalConfalse}
83 \DeclareOption{contents}{\xweb_GlobalContrue}
84 \DeclareOption{hideformats}{\xweb_GlobalHideFormatstrue}
85 \DeclareOption{showformats}{\xweb_GlobalHideFormatsfalse}
86
87 \newcount\xweb_hypertype
88 \newcount\xweb_GlobalHypertype
89 \DeclareOption{hyperref}{\xweb_GlobalHypertype=1}
90 \DeclareOption{pdftex}{\xweb_GlobalHypertype=2}
91 \DeclareOption{nohype}{\xweb_GlobalHypertype=0}
92
93 \newcounter{xwebLopDepth} \setcounter{xwebLopDepth}{9}
94 \newcounter{xwebOutlineDepth} \setcounter{xwebOutlineDepth}{9}
95 \DeclareOption{nofiles}{\c@xwebLopDepth=-9}
96 \DeclareOption{writefiles}{\c@xwebLopDepth=9}
97 \ExecuteOptions{index,reflist,raggedbottom,allsections,%
98               contents,showformats,nohype,writefiles}
99 </main>

```

8 The interface between cweave and T_EX.

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.

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

`\ATL` does only appear in front of the very first section. The section number is an explicit T_EX 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 T_EX 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:

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

⁴This 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.

<code>\.</code>	1	C string <i>Arg. 1:</i> string
<code>\)</code>		discretionary break between string parts
<code>\&</code>	1	reserved identifier <i>Arg. 1:</i> identifier
<code>\\</code>	1	“normal” identifier with more than one chars <i>Arg. 1:</i> identifier
<code>\ </code>	1	“normal” identifier with one char <i>Arg. 1:</i> character
<code>\C</code>	1	C comment <i>Arg. 1:</i> comment text
<code>\MRL</code>	1	C operator: combined binary operators <i>Arg. 1:</i> operators, <code>\K</code> must print as ‘=’
<code>\SHC</code>	1	C++ comment <i>Arg. 1:</i> comment text
<code>\T</code>	1	numeric constants <i>Arg. 1:</i> constant
<code>\X</code>	<code>#1: #2\X</code>	refinement name <i>Arg. 1:</i> section number <i>Arg. 2:</i> 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:

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

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

<code>\1</code>	future stmts indented one more unit
<code>\2</code>	future stmts indented one less unit
<code>\3</code>	1 optional line break within a statement <i>Arg. 1:</i> digit, penalty for line break
<code>\4</code>	backspace one indentation unit
<code>\5</code>	optional line break or small space (between run-in statements)
<code>\6</code>	line break
<code>\7</code>	line break and additional vertical space
<code>\8</code>	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:

<code>_</code>	space
<code>\&</code>	ampersand
<code>\\</code>	backslash
<code>\^</code>	hat
<code>_</code>	underscore
<code>\{</code>	left brace
<code>\}</code>	right brace
<code>\~</code>	tilde

8.6 Last, we have tags which are used in their “standard” L^AT_EX meaning. We don’t have much work with them:

<code>\#</code>	hash mark in program or in string
<code>\\$</code>	dollar
<code>\%</code>	percent
<code>\{</code>	block start (is set in math mode)
<code>\}</code>	block end (is set in math mode)
<code>\,</code>	thin space (like the L ^A T _E X default)
<code>\langle</code>	left delimiter in include directives, or in templates
<code>\le</code>	C operator: less or equal
<code>\ldots</code>	Function prototypes: Variable number of arguments
<code>\rangle</code>	right delimiter in include directives, or in templates

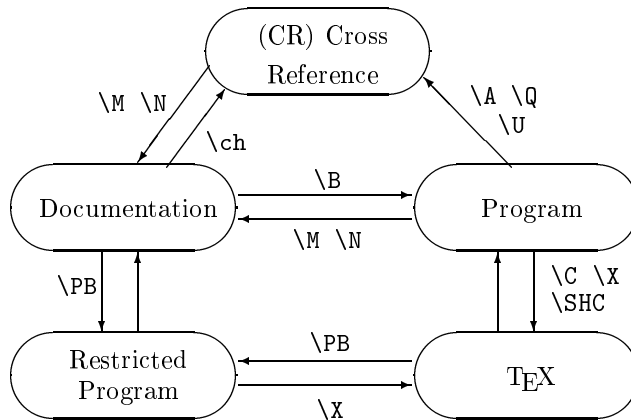


Figure 2: The processing state’s FSA. The automaton starts and ends in the “documentation” state.

9 Processing states.

We have to typeset five different categories of material: Documentation, program pieces—embedded within the documentation and as large chunks, $\text{T}_\text{E}\text{X}$ 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. $\backslash\text{B}$ switches to *program state*. This can happen in documentation and program state.
3. While we process the argument of $\backslash\text{PB}$ we’re in *restricted program state*; $\backslash\text{PB}$ may appear in documentation and in $\text{T}_\text{E}\text{X}$ state. Since $\text{T}_\text{E}\text{X}$ state can be switched on within restricted program state, $\backslash\text{PB}$ can appear within the argument of itself.
4. In the arguments of $\backslash\text{C}$, $\backslash\text{SHC}$, and $\backslash\text{X}$ we switch to *$\text{T}_\text{E}\text{X}$ 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.

$\backslash\text{xweb_documentation}$ will switch to documentation state, $\backslash\text{xweb_program}$ to program state, $\backslash\text{xweb_Rprogram}$ to restricted program state, $\backslash\text{xweb_tex}$ to $\text{T}_\text{E}\text{X}$ state, and $\backslash\text{xweb_CR}$ 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 $\text{T}_\text{E}\text{X}$ 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 $\text{T}_\text{E}\text{X}$ 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 $\text{T}_\text{E}\text{X}$ 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.

```

100 <*main>
101 \let\xweb_UserBindings=\relax
102 \let\xweb_DocLayout=\relax
103 \def\xweb_documentation{%
104   \xweb_UserBindings
105   \xweb_DocLayout}
106 \def\xweb_Rprogram{%
107   \xweb_CweaveBindings}
108 \def\xweb_program{%
109   \xweb_CweaveBindings
110   \xweb_ProgramLayout}
111 \def\xweb_tex{%
112   \xweb_UserBindings}
113 \def\xweb_CR{%
114   \xweb_CweaveBindings
115   \xweb_DocLayout}
116 </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 $(\backslash stop, \backslash stop)$. 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.

```

117 (*main)
118 \newtoks\xweb_undefined
119 \newtoks\xweb_ToRestore
120 \def\xweb_rebind{%
121   \xweb_undefined{}%
122   \xweb_ToRestore{}%
123   \xweb_DoRebind
124 }
125 \def\xweb_DoRebind#1#2{%
126   \ifx #1\stop
127     \let\next\relax
128   \else
129     \xweb_SaveBinding #2%
130     \let #2=#1%
131     \let\next\xweb_DoRebind
132   \fi
133   \next
134 }
135 /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.”

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

```

136 (*main)
137 \def\xweb_SaveBinding#1{%
138   \ifx #1\undefined
139     \xweb_undefined \expandafter{\the\xweb_undefined #1}%
140   \else
141     \expandafter\let \csname xweb_s\_string#1\endcsname =#1%

```

```

142     \xweb_ToRestore \expandafter{\the\xweb_ToRestore #1}%
143     \fi
144   }
145 </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.

```

146 (*main)
147 \def\xweb_RestoreBindings{%
148   \expandafter\xweb_undefine \the\xweb_undefined \stop
149   \xweb_undefined{%
150     \expandafter\xweb_RestoreBinding \the\xweb_ToRestore \stop
151     \xweb_ToRestore{%
152     }
153 \def\xweb_undefine#1{%
154   \ifx #1\stop
155     \let\next\relax
156   \else
157     \let#1\undefined
158     \let\next\xweb_undefine
159   \fi
160   \next
161 }
162 </main>

```

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



Another `\next` assignment.

```

163 (*main)
164 \def\xweb_RestoreBinding#1{%
165   \ifx #1\stop
166     \let\next\relax
167   \else
168     \expandafter\let \expandafter#1\csname xweb\string#1\endcsname
169     \expandafter\let \csname xweb_s_\string#1\endcsname \undefined
170     \let\next\xweb_RestoreBinding
171   \fi
172   \next
173 }
174 </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.

```

175 ⟨*main⟩
176 \def\xweb_anchor{xwebmod:\xwebJobname:\arabic{xwebModule}}
177 \def\xweb_SetupHrefs{
178   \ifnum\xweb_hypertype=0
179     \def\xweb_hlink##1{##1}
180     \def\xweb_htarget##1{##1}
181   \else\ifnum\xweb_hypertype=1
182     \def\xweb_hlink##1{\hyperlink{xweb_anchor}{##1}}
183     \def\xweb_htarget##1{\hypertarget{xweb_anchor}{##1}}
184   \else\ifnum\xweb_hypertype=2
185     \def\xweb_hlink##1{\xweb_pdflink{xweb_anchor}{##1}}
186     \def\xweb_htarget##1{\xweb_pdftarget{xweb_anchor}{##1}}
187   \else\PackageError{webfiles}{This can't happen.}{%
188     this may be a bug in the webfiles package}
189   \fi\fi\fi}
190 \xweb_SetupHrefs
191 ⟨/main⟩

```

12 PDF_TE_X support.

PDF_TE_X is an implementation of T_EX 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.

```

192 (*main)
193 \def\xwebPDFOutline#1#2{%
194     \edef\tmpnr{#1}%
195     \edef\name{#2}%
196     \c@xwebModule=\tmpnr\relax
197     \ifnum\xweb_hypertype=2
198         \edef\anchor{\xweb_anchor}
199         \pdfoutline goto name{\anchor}{#1 \name}
200     \fi}}
201 (/main)

```

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

```

202 (*main)
203 \def\xweb_pdflink#1#2{%
204     \pdfannotlink
205         attr{/Border [0 0 0]}
206         goto name{#1}
207         \Red #2\Black
208     \pdfendlink}
209 \def\xweb_pdftarget#1#2{
210     \pdfdest name{#1} fitbh
211     #2}
212 (/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.

```
213 (*main)
214 \def\xwebLapStar{\rlap{*}}
215 (/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.

```
216 (*main)
217 \newcount\xwebGroupLevel
218 \newcounter{xwebSecNoEject}
219     \setcounter{xwebSecNoEject}{3}
220 (/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 whether 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_PrepereSection`, which is used before each section. This macro also changes to documentation state.

```
221 (*main)
222 \newcounter{xwebModule}
223 \newif\ifon % determines if the section is 'on'
224 \def\onmaybe{\let\ifon=\maybe} % analogous to \ontrue
225 \let\maybe=\iftrue % initial meaning; the user can say "\let\maybe=\iffalse"
226 \def\xweb_PrepereSection#1{%
227     {\xdef\secstar{#1}%
228     \let*=\empty% remove \* from the number
229     \xdef\xweb_secno{#1}}
230     \ifx\xweb_secno\secstar % if they are the same; i.e.:
231         \onmaybe           % if there was no star in #1
232     \xdef\xweb_star{}%
233     \else
234         \ontrue
235     \xdef\xweb_star{*}%
236     \fi
237     \setcounter{xwebModule}{#1}% global set!
238     \xweb_documentation
239 }
240 (/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 `\unvboxed` 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 `unboxed`, 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.

```

241 (*main)
242 \def\thexwebModule{\arabic{xwebModule}}
243 \newbox\xweb_ModBox
xwebModule 244 \newif\ifxweb_FirstModule \xweb_FirstModulefalse
245 \newenvironment{xwebModule}{%
246   \ifxweb_FirstModule
247     \ifxwebCon\xwebContents{\xwebJobname}\fi
248     \global\xweb_FirstModulefalse
249     \openout\xweb_cont=\xwebJobname.con
250     \write\xweb_cont{\catcode '\noexpand@=11\relax}%
251   \fi
252   \ifxwebRagged\vfil\penalty-200\vfilneg% (TeXbook p.353)
253     % (this is a \filbreak without a \par)
254   \else\penalty-200\fi
255   %\setbox\xweb_ModBox=\vbox\bgroup
256   \edef\@currentlabel{\thexwebModule}
257   \par\bigskip
258   \everypar{%
259     \if@noskipsec%
260       \global\@noskipsecfalse%
261       \everypar{\penalty-100}%
262     \fi}
263   \@noskipsectrue
264   \mbox{} \hspace{-\parindent}%
265   \xwebModuleHook
266   \xweb_htarget{\textbf{\thexwebModule.}\xweb_star}% typeset module number
267   \hskip 1em plus0.1em minus 0.1em%
268   }{\% \egroup \unvbox \xweb_ModBox
269   }
270 \def\xwebModuleHook{}
271 (/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 L^AT_EX 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).

```

272 (*main)
273 \def\xwebMainSecSkip{\clearpage}
274 \def\N#1#2#3.{%
275   \ifon\end{xwebModule}\fi
276   \global\xwebGroupLevel #1           % <-- space!!
277   { \let\*=\empty%
278     \xdef\xweb_secno{#2}% get the section number
279   }%
280   \ifnum\xweb_secno>1
281     \ifnum \xwebGroupLevel<\c@xwebSecNoEject
282       \xwebMainSecSkip
283     \fi
284   \fi
285   \message \expandafter{*}\xweb_secno}%
286   \xweb_PrepareSection{#2}
287   \ifon
288     \def\xweb_ModTitle{#3}%
289     \begin{xwebModule}{\textbf{#3.}}% title
290     \hskip 1em plus.1em minus.1em%
291     \ifnum\c@xwebLopDepth > #1%
292       \addcontentsline{lop}{starred}{%
293         \protect\global\xwebGroupLevel #1 % space after #1 !
294         {\thexwebModule.}~{#3}}%
295       { \let\protect\@unexpandable@protect
296         \edef\next{\write\xweb_cont{%
297           \ZZ{#3}{#1}{\xweb_secno}{\thepage}}}%
298         \next
299       }% write "\ZZ{title}{depth}{sec}{page}" to .con file
300     \fi
301     \ifnum\c@xwebOutlineDepth > #1%
302       \xwebPDFOutline{\xweb_secno}{#3}% args: nr, title
303     \fi
304 }
305 (/main)

```

13.5 Normal sections are started with `\M`.

```

306 (*main)
307 \def\M#1{%
308   \ifon\end{xwebModule}\fi
309   \xweb_PrepareSection{#1}%
310   \ifon\begin{xwebModule}%
311     \ifnum\c@xwebLopDepth>9%
312       \addcontentsline{lop}{xwebsection}{%

```

```
313     \protect\global\xwebGroupLevel 4 \thexwebModule.}%
314   { \let\protect\@unexpandable@protect
315     \edef\next{\write\xweb_cont{%
316       \ZZ{}{4}{#1}{\thepage}}}%
317     \next
318   }% write "\ZZ{title}{depth}{sec}{page}" to .con file
319   \fi
320 }
321 (/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`.

```

322 (*main)
323 \def\xwebTocName{Contents of \xwebJobname}
324 \def\xwebLopName{List of Programs}
325 \ifx\listofprograms\undefined
326   \ifx \chapter\undefined
327     \def\listofprograms{\section*{\xwebLopName\@mkboth
328       {\uppercase{\xwebLopName}}{\uppercase{\xwebLopName}}}\@starttoc{lop}}
329   \else
330     \newcommand{\listofprograms}{\chapter*{\xwebLopName\@mkboth
331       {\uppercase{\xwebLopName}}{\uppercase{\xwebLopName}}}\@starttoc{lop}}
332   \fi
333 \fi
334 (/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:

```

335 (*main)
336 \def\l@program#1{%
337   \xwebGroupLevel\z@           % default value of group level
338   \begingroup
339     \let\numberline\@gobble     % width not known yet
340     \setbox\z@ \hbox{#1}%
341   \endgroup
342   \xweb_TTLopLine{#1}%
343 }
344 (/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.

```

345 (*main)
346 \def\l@xwebSection#1{
347   \xwebGroupLevel\z@ % default value of group level
348   \begingroup
349     \let\numberline\@gobble     % width not known yet
350     \setbox\z@ \hbox{#1}%
351   \endgroup
352   \xweb_NormalTocLine{#1}%
353 }
354 (/main)

```


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

```

389 <*main>
390 \def\xweb_BoldTocLine#1{%
391   \@dottedtocline
392     1                               % level
393     {\xwebGroupLevel\xwebLopIndent}% % basic indent
394     \xwebLopIndent                 % numwidth
395     {\bf #1}}%                     % entry
396   }
397 </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.

```

398 <*main>
399 \def\xweb_TTLopLine#1#2{%
400   \addpenalty{\@secpenalty}%
401   \addvspace{2em plus\p@}%
402   \begingroup
403     \noindent
404     \hangindent\xwebLopIndent
405     \rightskip\@tocrmarg \parfillskip -\rightskip
406     \interlinepenalty\@M
407     \@tempdima\xwebLopIndent      % for \numberline
408     {\large\bf\tt #1}\nobreak\hfill \hbox to\@pnumwidth{\hss #2}%
409     \par
410   \endgroup
411   \addvspace{.5em plus\p@}%
412   }
413 </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.

```

414 (*main)
415 \def\PB#1{%
416   \begingroup
417     \xweb_Rprogram
418     \leavevmode
419     #1%
420   \endgroup}
421 (/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.

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

```

422 (*main)
423 \newcount\xwebProgPenalty
424   \xwebProgPenalty=\@secpenalty
425   \divide\xwebProgPenalty by 2
426 \def\Y{%
427   \if@noskipsec
428   \else
429     \par
430     \penalty\xwebProgPenalty
431     \smallskip
432   \fi}
433 (/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.

```

434 (*main)
435 \def\xweb_CweaveBindings{%
436   \ifx \xweb_UserBindings\relax
437     \xweb_rebind
438     % indentation and paragraph layout
439     \xweb_IncrIndent   \1%
440     \xweb_DecrIndent   \2%
441     \xweb_ExprBreak    \3%
442     \xweb_backup       \4%
443     \xweb_OptBreak     \5%
444     \xweb_break        \6%
445     \xweb_BigBreak     \7%

```

```

446      \xweb_noindent      \8%
447      % C/C++ tokens
448      \xwebRel            \?%
449      \xwebAddress       \AND
450      \xwebComplement    \CM
451      \xwebScope         \DC
452      \xwebEquiv         \E
453      \xwebGe            \G
454      \xwebRightShift    \GG
455      \xwebNe            \I
456      \xwebAssign        \K
457      \xwebLeftShift     \LL
458      \xwebMod           \MOD
459      \xwebNull          \NULL
460      \xwebNot           \R
461      \xwebBinOr         \OR
462      \xwebMemberRef     \PA
463      \xwebThis          \this
464      \xwebOr            \V
465      \xwebAnd           \W
466      \xwebXor           \XOR
467      \xwebLE            \Z
468      \xwebPointer       \MG
469      \xwebPointerMemberRef \MGA
470      \xwebDecr          \MM
471      \xwebIncr          \PP
472      % more tokens
473      \xwebId             \\\%
474      \xwebIdLetter      \\\%
475      \xwebRes           \&%
476      \xwebString        \.%           %% ( ...Emacs...
477      \xwebStringBreak   \)\%
478      \xwebNumber        \T
479      \xwebCombinedOp    \MRL
480      % goes to TeX state
481      \xwebComment       \C
482      \xwebCxxComment    \SHC
483      \xwebRefName       \X
484      % CWEB tokens
485      \xwebMacrosHere    \ATH
486      \xwebDefine        \D
487      \xwebFormat        \F
488      \xwebIdCat         \J
489      \xwebVerbString    \vb
490      % cross reference tags
491      \xwebChangeFlag    \*%
492      \xwebCRAlso        \A
493      \xwebCRsAlso       \As
494      \xwebCRCite        \Q
495      \xwebCRsCite       \Qs
496      \xwebCRUse         \U
497      \xwebCRsUse        \Us
498      \xwebCREt          \ET
499      \xwebCRsEt         \ETs
500      % finish the list
501      \stop\stop
502      \def\xweb_UserBindings{%
503      \xweb_RestoreBindings
504      \let\xweb_UserBindings\relax
505      }%
506      \fi
507      }

```

508 `/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.

```

509 (*main)
510 \newskip\xweb_SaveParskip
511 \newskip\xweb_SaveRightskip
512 \newcount\xweb_SaveSemSFCode
513 \newcount\xweb_SavePretolerance
514 \newcount\xweb_SaveHyphenpenalty
515 \newcount\xweb_SaveExhyphenpenalty
516
517 \def\xweb_SaveDocLayout{%
518   \xweb_SaveParskip\parskip
519   \xweb_SaveRightskip\rightright
520   \xweb_SaveSemSFCode\sfcodes; % changed by \xweb_ProgramLayout
521   \xweb_SavePretolerance\pretolerance
522   \xweb_SaveHyphenpenalty\hyphenpenalty
523   \xweb_SaveExhyphenpenalty\exhyphenpenalty
524 }
525 \def\xweb_RestoreDocLayout{%
526   \parskip\xweb_SaveParskip
527   \rightskip\xweb_SaveRightskip
528   \sfcodes;=\xweb_SaveSemSFCode
529   \pretolerance\xweb_SavePretolerance
530   \hyphenpenalty\xweb_SaveHyphenpenalty
531   \exhyphenpenalty\xweb_SaveExhyphenpenalty
532 }
533 /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`.

```

534 (*main)
535 \def\xweb_ProgramLayout{%
536   \ifx \xweb_DocLayout\relax
537     \xweb_LayoutInit
538     \xweb_SaveDocLayout
539     \def\xweb_DocLayout{%
540       \xweb_RestoreDocLayout
541       \let\xweb_DocLayout\relax
542     }%
543     % set new values
544     \parskip\z@skip
545     \rightskip\z@ plus 100\p@ minus 10\p@
546     \sfcodes;=3000 % same stretch factor as period.
547     \pretolerance\@M
548     \hyphenpenalty 9999 % strings can be broken this way
549     \exhyphenpenalty\@M
550   \fi
551 }
552 /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.

```

553 (*main)
554 \newdimen\xwebIndentUnit   \xwebIndentUnit=1em
555 \newdimen\xweb_indent
556 \newbox\xweb_bak           % backspace one unit
557 \newbox\xweb_bakk          % backspace two units
558 \def\xweb_LayoutInit{%
559   \global\setbox\xweb_bak \hbox to -1\xwebIndentUnit{}%
560   \global\setbox\xweb_bakk \hbox to -2\xwebIndentUnit{}%
561   \global\let\xweb_LayoutInit\relax
562 }
563 (/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.

```

564 (*main)
565 \def\B{%
566   \if@noskipsec \hskip 1em \fi
567   \xweb_program
568   \xweb_indent 3\xwebIndentUnit \hangindent\xweb_indent
569   \ifvmode
570     \if@noskipsec
571       \indent           % add an empty documentation part
572     \else
573       \noindent \kern\xwebIndentUnit
574     \fi
575   \fi
576 }
577 (/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.

```

578 (*main)
579 \def\xweb_break{%           % forced break, between statements
580   \ifmode\else
581     \endgraf               % in LaTeX it isn't sure what \par is really...
582     \noindent
583     \hangindent\xweb_indent \kern\xweb_indent
584     \copy\xweb_bakk % go back to basic indentation
585     \ignorespaces
586   \fi
587 }

```

```

588 \def\xweb_OptBreak{%      % optional break between statements
589     \hfil \penalty\m@ne
590     \hfilneg \kern .5em \kern 2\xwebIndentUnit    % discarded on line break
591     \copy\xweb_bakk
592     \ignorespaces
593     }
594 \def\xweb_BigBreak{%      % forced break and a little extra space
595     \Y
596     \xweb_break
597     }
598 \def\xweb_ExprBreak#1{% % break with penalty #1 * 10
599     \hfil \penalty#10
600     \hfilneg          % discarded on line break
601     }
602 (/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?

```

603 (*main)
604 \def\xweb_IncrIndent{%
605     \global\advance\xweb_indent\xwebIndentUnit
606     \hangindent\xweb_indent
607     }
608 \def\xweb_DecrIndent{\global\advance\xweb_indent -\xwebIndentUnit}
609 \def\xweb_backup{\copy\xweb_bak}
610 \def\xweb_noindent{%          % no indentation
611     \hskip -\xweb_indent \hskip 2\xwebIndentUnit
612     }
613 (/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.

```

614 ⟨*main⟩
615 \let\xwebAnd=\land           % logical and, &&
616 \let\xwebEquiv=\equiv       % equiv sign, for ==
617 \let\xwebGe=\ge             % greater or equal
618 \let\xwebLE=\le             % less or equal
619 \let\xwebNe=\ne             % unequal, !=
620 \let\xwebNull=\Lambda       % NULL pointer
621 \let\xwebNot=\lnot           % logical not, !
622 \let\xwebOr=\lor             % logical or, ||
623 \let\xwebXor=\oplus         % bitwise exclusive or, ^
624 ⟨/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 T_EX 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 L^AT_EX 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.

```

625 ⟨*main⟩
626 \font\teni=cmmi10
627 \font\sevensy=cmsy7
628
629 \newbox\xweb_pointer         % pointer to struct component, ->
630 \setbox\xweb_pointer=\hbox{\kern-2pt\lower3pt\hbox{\teni\char'176}\kern1pt}
631
632 \newbox\xweb_decr           % decrement, --
633 \setbox\xweb_decr=\hbox{%
634     \kern .5pt
635     \raise 1pt \hbox{\sevensy\char0 \kern-1pt\char0}%
636     \kern .5pt
637 }
638 \newbox\xweb_incr           % increment, ++
639 \setbox\xweb_incr=\hbox{%
640     \kern .05em
641     \raise .1em \hbox{\scriptstyle {+}\kern -.1em{+}}%
642     \kern .05em
643 }
644 ⟨/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...


```

645 (*main)
646 \mathchardef\xwebAddress="2026 % '&', as binary op
647 \let\xwebAssign== % assignment
648 \let\xwebBinOr=\mid % binary or
649 \def\xwebComplement{\sim} % '~', as ordinary symbol
650 \def\xwebDecr{\copy\xweb_decr} % decrement
651 \def\xwebIncr{\copy\xweb_incr} % increment
652 \let\xwebLeftShift=\ll % left shift, <<
653 \def\xwebMod{\mathbin{\hbox{\footnotesize\rm\%}}} % modulo/remainder, %
654 \def\xwebMemberRef{\mathbin{.}} % ptr to member (on object)
655 \def\xwebPointer{\copy\xweb_pointer}
656 \def\xwebPointerMemberRef{\mathbin{\xwebPointer*}} % ptr to member (on ptr)
657 \def\xwebRel{\mathrel?} % relation operator
658 \let\xwebRightShift=\gg % right shift, >>
659 \def\xwebScope{\kern.1em{:}\kern.1em} % scope resolution
660
661 \def\xwebThis{\xwebRes{this}} % reserved identifier 'this'
662 </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?

```

663 (*main)
664 \def\xwebCombinedOp#1{\mathrel{\let\K==#1}} % eg, += operator
665 \def\xwebId#1{\hbox{\it#1/\kern.05em}} % identifier, more than one char
666 \def\xwebIdLetter#1{\hbox{#1$}} % identifier, one letter
667 \def\xwebRes#1{%
668   \hbox{\bf
669     \def_{\kern.04em\vbox{\hrule width.3em height .6pt}\kern.08em}%
670     #1/\kern .05em
671   }%
672 }
673 </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 be 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.

```

674 (*main)
675 \def\xwebString#1{%
676   \hbox{\tt
677     \xweb_StringSetup
678     #1\kern .05em
679   }%



```

```

680   }
681 \def\xweb_StringSetup{%
682   \chardef\ ='\ %           % <-- two spaces !
683   \chardef\&='\&
684   \chardef\=\='\
685   \chardef\^='\^
686   \chardef\_='\_
687   \chardef\{='\{
688   \chardef\}='\}
689   \chardef\~='\~
690   \def\{'\'}%
691   }
692 \def\xwebStringBreak{\discretionary{\hbox{\tt\char'\}}{\}}
693 \}/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.

```

694 (*main)
695 \def\xweb_oct{\hbox{\$^\circ$\kern-.1em\it\aftergroup\?\aftergroup}}
696 \def\xweb_hex{\hbox{\$^\scriptscriptstyle\#\}$\tt\aftergroup}}
697
698 \def\xwebNumber#1{%           % octal, hex, or decimal constant
699   \hbox{%
700     $%
701     \def\?{\kern.2em}%
702     \def\###1{\egroup\sb{\,\rm##1}\bgroup}% suffix to constant
703     \def\_ {\cdot 10^{\aftergroup}}% power of ten (via dirty trick)
704     \let~\xweb_oct \let^~\xweb_hex
705     {#1}%
706     $}%
707   }
708 \}/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.

```

709 (*main)
710 \def\xwebComment#1{%
711   \5%           % 0.5em will be discarded on line break
712   \hskip 1.5em
713   $\ast\,$%
714   {\xweb_tex
715     \xwebCommentHook
716     #1%

```

```
717     }%
718     $\\,\\ast/$%
719     }
720 \\let\\xwebCommentHook\\relax
721 \\let\\xwebCxxComment\\xwebComment
722 </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`).

```

723 ⟨*main⟩
724 \def\xwebIdCat{\xwebString{@&}}
725 \def\xwebMacrosHere{%
726   \begingroup
727     \def\xwebRefNumber##1{}%
728     \xwebRefName :\xwebRefMacrosHere\X
729   \endgroup
730 }
731 \def\xwebRefMacrosHere{Preprocessor Definitions}
732 ⟨/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.

```

733 ⟨*main⟩
734 \def\xwebVerbString#1{\fboxsep\tw@p@ \fbox{\xwebString{#1}}}
735 ⟨/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`.

T_EXnicl 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?

```

736 (*main)
737 \def\xwebRefName#1:#2\X{%
738   \ifmmode \gdef\xweb_ToggleText{\null$\null}%
739   \else \let\xweb_ToggleText\relax
740   \fi
741   \xweb_ToggleText
742   $\langle\$, {\xweb_tex \xweb_CheckDot{#2}\xwebRefNumber{#1}}$\rangle\$, \rangle$%
743   \xweb_ToggleText
744   }
745 (/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.

```

746 (*main)
747 \def\xwebRefNumber#1{%
748   {\reset@font \footnotesize \kern .5em%
749     \xwebSetModNrList#1.}}
750 (/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.

```

751 (*main)
752 \def\xwebSetModuleNumber#1{{\edef\tmpnr{#1}%
753   \c@xwebModule=\tmpnr\relax
754   \xweb_hlink{\thexwebModule}}}
755 (/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.

```

756 (*main)
757 \newtoks\xweb_toksA
758 \newtoks\xweb_toksB
759 \newtoks\xweb_toksC
760 \newtoks\xweb_toksD
761 \newcount\xweb_countA
762 \xweb_countA=0
763 \def\xweb_AddTokens#1#2{%
764   \edef\addtoks{\noexpand#1={\the#1#2}}\addtoks}
765 \def\xweb_adn#1{\xweb_AddTokens{\xweb_toksC}{#1}%
766   \global\xweb_countA=1\let\next=\maketoks}
767 \def\xweb_poptoks#1#2|ENDTOKS|{%
768   \let\first=#1\xweb_toksD={#1}\xweb_toksA={#2}}
769 \def\maketoks{%
770   \expandafter\xweb_poptoks\the\xweb_toksA|ENDTOKS|
771   \ifx\first0\xweb_adn0

```



```

772 \else\ifx\first1\xweb_adn1 \else\ifx\first2\xweb_adn2
773 \else\ifx\first3\xweb_adn3 \else\ifx\first4\xweb_adn4
774 \else\ifx\first5\xweb_adn5 \else\ifx\first6\xweb_adn6
775 \else\ifx\first7\xweb_adn7 \else\ifx\first8\xweb_adn8
776 \else\ifx\first9\xweb_adn9
777 \else
778 \ifnum0=\xweb_countA\else\xweb_makenote\fi
779 \ifx\first.\let\next=\xweb_tdone\else
780 \let\next=\maketoks
781 \xweb_AddTokens{\xweb_toksB}{\the\xweb_toksD}
782 \ifx\first,\xweb_AddTokens{\xweb_toksB}{\space}\fi
783 \fi
784 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
785 \next
786 }
787 \def\xweb_makenote{\xweb_AddTokens{\xweb_toksB}%
788 {\noexpand\xweb_TagSec{\the\xweb_toksC}}%
789 \xweb_toksC={}\global\xweb_countA=0}
790 \def\xweb_tdone{\edef\st{\global\noexpand\xweb_toksA={\the\xweb_toksB}}\st}
791 \end{main}

```

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

```

792 \begin{main}
793 \def\xweb_TagSec#1{\xwebSetModuleNumber{#1}}
794 \def\xwebSetModNrList#1.{%
795 \setbox0=\hbox{\xweb_toksA={#1.}\xweb_toksB={}\maketoks}%
796 \the\xweb_toksA}
797 \end{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`.

```

798 \begin{main}
799 \def\xweb_CheckDot#1{%
800 \begin{group}
801 \let\.\@gobble
802 \let\PB\relax
803 \expandafter\def \expandafter\next \expandafter{#1}%
804 \ifx \next\empty
805 \gdef\next##1{\xwebString}%

```

```

806     \else
807     \global\let\next\relax
808     \fi
809 \endgroup
810 \next #1
811 }
812 </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 introductory identifier (either `#define` or `format`) to set at the very front we increase the indentation by three units.

```

813 (*main)
814 \def\xwebDefine{\xweb_macro{\#define}}
815 \def\xwebFormat\#1\#2\par{\ifxwebHideFormats\else
816 \xweb_macro{format}\#{#1} \#{#2}\par\fi}
817 \def\xweb_macro#1{%
818 \global\advance\xweb_indent \tw@\xwebIndentUnit
819 \xweb_IncrIndent
820 \xwebRes{#1 }%           % <-- blank!
821 }
822 </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 introductory 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 auxiliary, 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 introductory 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.

```
823 (*main)
824 \def\xwebNumberListHangindent{2em}
825 \def\xwebCrossRef#1#2.{%
826   \par\smallskip% (= \Y)
827   \xweb_CR%
828   \begingroup%
829     \reset@font \footnotesize%
830     \noindent \hangindent\xwebNumberListHangindent%
831     #1~\xwebSetModNrList #2..\par%
832   \endgroup%
833 }%
834 (/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`.

```
835 (*main)
836 \def\xwebCRAlso{\xwebCrossRef{See also section}}
837 \def\xwebCRsAlso{\xwebCrossRef{See also sections}}
838
839 \def\xwebCRCite{\xwebCrossRef{This code is cited in section}}
840 \def\xwebCRsCite{\xwebCrossRef{This code is cited in sections}}
841
842 \def\xwebCRUse{\xwebCrossRef{This code is used in section}}
843 \def\xwebCRsUse{\xwebCrossRef{This code is used in sections}}
844 (/main)
```

18.3 The `changeFlag` is set as a star.

```
845 (*main)
846 \let\xwebChangeFlag=*
847 \def\xwebCRET{ and~}
848 \def\xwebCRsEt{, and~}
849 (/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_extEntries` (where $ext \in \{\text{idx}, \text{scn}\}$ is the extension of the appropriate file) which tells if there are any entries of the respective category.

```
850 (*main)
851 \newif\ifxweb_idxEntries
852 \newif\ifxweb_scnEntries
853 (/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.

```
854 (*main)
855 \newread\xweb_TmpFile
856 \def\xweb_HasEntries#1{%
857   \openin\xweb_TmpFile \xwebJobname.#1\relax
858   \ifeof \xweb_TmpFile
859     \let\next\@defpar
860     \message{No #1 file present for \xwebJobname !}
861   \else
862     \read\xweb_TmpFile to \next
863   \fi
864   \csname xweb_#1Entries%
865     \ifx \next\@defpar false%
866     \else true%
867   \fi
868   \endcsname
869   \closein\xweb_TmpFile
870 }
871 (/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.

```
872 (*main)
873 \newwrite\xweb_cont
874 \let\ZZ=\let % now you can \write the control sequence \ZZ
875 \def\xweb_contentsline#1#2#3#4{%
876   \ifnum#2=0 \smallbreak\fi
877   \hbox to \textwidth{\xweb_consetup{#2}#1
```

```

878     \rm\leaders\hbox to .5em{.\hfil}\hfil\ %
879     \xwebSetModuleNumber{#3}
880     \hbox to3em{\hss#4}}
881 \def\xweb_consetup#1{\ifcase#1 \bf % depth -1 (@**)
882  \or % depth 0 (@*)
883  \or \hskip2em % depth 1 (@*1)
884  \or \hskip4em % depth 2 (@*2)
885  \or \hskip6em % depth 3 (@*3)
886  \or \hskip8em % depth 4 (@*4)
887  \or \hskip10em % depth 5 (@*5)
888  \else \hskip12em \fi} % depth 6 or more
889 \endgroup

```

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

```

890 (*main)
891 \def\xwebContents#1{
892   \def\readcontents{\IfFileExists{#1.con}{\input #1.con}{}}
893   \begingroup
894     \xwebContentsTop
895     \ifnum\xweb_hypertype=2
896       \pdfdest name{#1:contents} fith
897       \pdfoutline goto name{#1:contents}{#1}
898     \fi
899     \hbox to \textwidth{\hfil
900       \xwebSectionName\hbox to3em{\hss \xwebPageName}}
901     \let\ZZ=\xweb_contentsline
902     \readcontents\relax % read the contents info
903     \xwebContentsBot % print the contents page(s) and terminate
904   \endgroup
905 }
906 \def\xwebSectionName{Section}
907 \def\xwebPageName{Page}
908 \endgroup

```

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 multiple 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).

```

909 (*main)
910 \def\xwebContentsTop{\vskip.2in}
911 \def\xwebContentsBot{\par\bigskip}
912 \def\xweb_InputWebfile#1{
913   \def\setname##1.##2.##3/{\gdef\xwebJobname{##1}}
914   \setname #1../
915   \ifnum\c@xwebLopDepth > -10 % entry to the list of programs:
916     \addcontentsline{lop}{program}{\xwebJobname}
917   \fi
918   \def\xwebContentsTop{\centerline{\textbf{\xwebJobname}}\vskip.2in}

```

```

919 \def\xwebContentsBot{\par\bigskip}
920 \global\xweb_FirstModuletrue
921 \begingroup
922 \let\inx\relax
923 \let\fin\relax
924 \let\con\relax
925 \let\ch\xwebCRChanged
926 \input{#1}
927 \ifon \end{xwebModule} \fi % end the last module (if it is on).
928 \endgroup
929 % process the index and list of refinements ...
930 \ifxwebIndex\xweb_HasEntries{idx}%
931 \ifxweb_idxEntries \xwebIdIndex{\xwebJobname.idx}\fi\fi
932 \ifxwebRef\xweb_HasEntries{scn}%
933 \ifxweb_scnEntries \xwebRefList{\xwebJobname.scn}\fi\fi
934 % clean up:
935 \closeout\xweb_cont
936 \onfalse % don't end the last section when starting the first section of
937 % the next web.
938 \xweb_UserBindings % back to the main document
939 }
940 </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 and including the result. This can be particularly worthwhile 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 refist outside the context of a web document.

```

941 (*main)
942 \def\xwebInputIndex#1{{
943 \def\setname##1.##2\{\%
944 \edef\xwebJobname{##1}
945 \ifx\##2\%
946 \edef\ext{idx}
947 \else
948 \edef\ext{\chop##2\}%
949 \fi}
950 \def\chop##1.\{\##1}
951 \expandafter\setname#1.\
952 \xwebIdIndex{\xwebJobname.\ext}
953 }}
954 \def\xwebInputRefList#1{{
955 \def\setname##1.##2\{\%
956 \edef\xwebJobname{##1}
957 \ifx\##2\%
958 \edef\ext{scn}
959 \else
960 \edef\ext{\chop##2\}%
961 \fi}
962 \def\chop##1.\{\##1}
963 \expandafter\setname#1.\
964 \xwebRefList{\xwebJobname.\ext}
965 }}
966 </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!

```

967 <*main>
968 \def\ifstr#1#2#3{%
969   \edef\xweb_tempa{#1}%
970   \edef\xweb_tempb{#2}%
971   \ifx\xweb_tempa\xweb_tempb #3\fi
972 }
973 </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.

```

974 <*main>
975 \def\webfile{\@ifnextchar[%
976   {\xweb_file}{\xweb_file[]}}
977 </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).

```

978 <*main>
979 \newif\if@opt
980 \def\xweb_off{\let\maybe=\iffalse}
981 \def\xweb_on{\let\maybe=\iftrue}
982 \def\xweb_file[#1]#2{
983   \ifxweb_GlobalIndex\xwebIndextrue\else\xwebIndexfalse\fi
984   \ifxweb_GlobalRef\xwebReftrue\else\xwebReffalse\fi
985   \ifxweb_GlobalRagged\xwebRaggedtrue\else\xwebRaggedfalse\fi
986   \ifxweb_GlobalOC\xweb_off\else\xweb_on\fi
987   \ifxweb_GlobalCon\xwebContrue\else\xwebConfalse\fi
988   \ifxweb_GlobalHideFormats\xwebHideFormatstrue\else
989     \xwebHideFormatsfalse\fi
990   \xweb_hypertype=\xweb_GlobalHypertype
991   \@for\xweb_opt:=#1\do{
992     \typeout{webfiles option: \xweb_opt}
993     \@optfalse
994     \ifstr{\xweb_opt}{index}{\xwebIndextrue\@opttrue}
995     \ifstr{\xweb_opt}{noindex}{\xwebIndexfalse\@opttrue}
996     \ifstr{\xweb_opt}{reflist}{\xwebReftrue\@opttrue}
997     \ifstr{\xweb_opt}{noreflist}{\xwebReffalse\@opttrue}
998     \ifstr{\xweb_opt}{raggedbottom}{\xwebRaggedtrue\@opttrue}
999     \ifstr{\xweb_opt}{flushbottom}{\xwebRaggedfalse\@opttrue}
1000    \ifstr{\xweb_opt}{onlychanges}{\xweb_off\@opttrue}
1001    \ifstr{\xweb_opt}{allsections}{\xweb_on\@opttrue}
1002    \ifstr{\xweb_opt}{nocon}{\xwebConfalse\@opttrue}
1003    \ifstr{\xweb_opt}{contents}{\xwebContrue\@opttrue}
1004    \ifstr{\xweb_opt}{hideformats}{\xwebHideFormatstrue\@opttrue}
1005    \ifstr{\xweb_opt}{showformats}{\xwebHideFormatsfalse\@opttrue}
1006    \ifstr{\xweb_opt}{hyperref}{\xweb_hypertype=1\@opttrue}
1007    \ifstr{\xweb_opt}{pdftex}{\xweb_hypertype=2\@opttrue}
1008    \ifstr{\xweb_opt}{nohype}{\xweb_hypertype=0\@opttrue}
1009    \if@opt\else\PackageError{webfiles}{%
1010      Unknown option for the \protect\webfile\space command

```

```

1011     }{Possible options are: index, noindex,
1012     reflist, noreflist, \MessageBreak
1013     raggedbottom, flushbottom, onlychanges,
1014     allsections, \MessageBreak
1015     nocon, contents, hideformats, showformats,
1016     hyperref, pdftex, nohype}
1017   \fi
1018 }
1019 \xweb_SetupHrefs
1020 \xweb_InputWebfile{#2}}
1021 \let\cwebfile\webfile
1022 \end{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 change flags any more—each section number in this list carries a change flag by definition.

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

```

1023 (*main)
1024 \def\xwebCRChanged{%
1025   \xwebCrossRef{The following sections were changed by the change file:}%
1026   \let*\relax
1027 }
1028 \end{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 introductory text for the index by `\xwebIndexIntro`. If there is any introductory 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.

```

1029 (*main)
1030 \newtoks\xwebIndexIntro
1031   \xwebIndexIntro={ }
1032 \def\xwebIndexName{Index of \texttt{\xwebJobname}}
1033 \end{main}

```

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

```

1034 (*main)
1035 \def\xweb_IndexTop{
1036   \xweb_tex
1037   \par\vskip 1cm plus 10mm minus 5mm\noindent
1038   {\bf \xwebIndexName}%\nopagebreak\par\medskip\nopagebreak
1039   %% skip already inserted by xweb_twocolumn env.
1040   \edef\intro{\the\xwebIndexIntro}% % is a local def
1041   \ifx \intro\empty
1042   \else
1043     \noindent\the\xwebIndexIntro\unskip
1044     \par\medskip

```



```

1045   \fi
1046 }
1047 </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.

```

1048 <*main>
1049 \newif\ifxweb_multicol \xweb_multicolfalse
1050 \IfFileExists{multicol.sty}{
1051   \if@twocolumn\else\xweb_multicoltrue\fi
1052 }{}
1053 \ifxweb_multicol
1054   \newenvironment{xweb_twocolumn}{\begin{multicols}{2}}{\end{multicols}}
1055 \else
1056   \newenvironment{xweb_twocolumn}{
1057     \if@twocolumn\else
1058       \columnseprule\z@
1059       \columnsep 35\p@           % value is from article.sty
1060     \fi
1061     \twocolumn%
1062   }{
1063     \onecolumn}
1064 \fi
1065 </main>

```

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



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

```

1066 <*main>
1067 \def\xwebJobName{noname}
1068 \def\xweb_SetupIndex{%
1069   \begin{xweb_twocolumn}[\xweb_IndexTop]
1070   \ifnum\xweb_hypertype=2
1071     \pdfdest name{\xwebJobname:index} fith
1072     \pdfoutline goto name{\xwebJobname:index}{Index}
1073   \fi
1074   % go to CR mode
1075   \xweb_CR
1076   \message{*index: }% % tell the user what we're doing
1077   % paragraph layout
1078   \begingroup
1079   \parskip \z@ plus .5\p@
1080   \parindent\z@
1081   \rightskip \z@ plus 2.5em
1082   \parfillskip \z@ plus .6\hspace
1083   \tolerance\@M \hyphenpenalty\@M
1084   % bindings
1085   \let\I\xwebIndexEntry
1086   \let\[\xwebIndexDeclared
1087   \let\*\xwebLapStar
1088 }
1089 \def\xweb_FinishIndex{
1090   \endgroup

```

```

1091   \end{xweb_twocolumn}
1092 }
1093 \def\xwebIdIndex#1{%
1094   \xweb_SetupIndex
1095   \@input{#1}
1096   \xweb_FinishIndex
1097 }
1098 (/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.)

```

1099 (*main)
1100 \def\xwebIndexEntry#1,#2.{%
1101   \par
1102   \hangindent\xwebNumberListHangindent
1103   \leavevmode
1104   \xweb_CheckDot{#1}:\quad\xwebSetModNrList #2.
1105 }
1106 (/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.

```

1107 (*main)
1108 \def\xwebIndexDeclared#1{\underline{#1}}
1109 \def\9#1{}
1110 (/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.

```

1111 (*main)
1112 \def\xwebReflistName{List of Refinements in \texttt{\xwebJobname}}
1113 \def\xwebRLCite{\xwebCrossRef{Cited in section}}
1114 \def\xwebRLsCite{\xwebCrossRef{Cited in sections}}
1115 \def\xwebRLUse{\xwebCrossRef{Used in section}}
1116 \def\xwebRLsUse{\xwebCrossRef{Used in sections}}
1117 (/main)

```

```

1118 (*main)
1119 \def\xweb_SetupReflist{%
1120   \vskip 3ex plus 7ex minus 1ex
1121   \xweb_tex
1122   \noindent
1123   {\bf \xwebReflistName}\nopagebreak\par\bigskip\nopagebreak
1124   \ifnum\xweb_hypertype=2
1125     \pdfdest name{\xwebJobname:reflist} fith
1126     \pdfoutline goto name{\xwebJobname:reflist} {Refs}
1127   \fi

```

```

1128 \xweb_CR
1129 \message{*list of refinements: }%
1130 % paragraph layout: like in index, but
1131 \parfillskip\@flushglue
1132 \begingroup % different bindings...
1133 \def\I{\par \hangindent\xwebNumberListHangindent}%
1134 \def\xwebCrossRef##1##2.{\quad {\reset@font\footnotesize
1135   ##1~\xwebSetModNrList ##2..}}%
1136 \def\Q {\xwebRLCite}%
1137 \def\Qs{\xwebRLsCite}%
1138 \def\U {\xwebRLUse}%
1139 \def\Us{\xwebRLsUse}%
1140 \let*\xwebChangeFlag
1141 }
1142 \def\xwebRefList#1{%
1143   \xweb_SetupReflist
1144   \@input{#1}
1145   \endgroup
1146 }
1147 \end{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.

```

1148 (*main)
1149 \def\ATL{\par\noindent\bgroup\catcode'\_ =12 \postATL} % print @1 in limbo
1150 \def\postATL#1 #2 {\bf letter \{\uppercase{\char"#1}}
1151   tangles as \tentex "#2"\egroup\par}
1152 \def\noATL#1 #2 {}
1153 \def\noat1{\let\ATL=\noATL} % suppress output from @1
1154 (/main)

```

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

```

1155 (*main)
1156 \ProcessOptions
1157 \ifxweb_multicol
1158   \RequirePackage{multicol}
1159 \fi
1160 \catcode'\_ =\xwebCatUsCode
1161 %\endinput
1162 (/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 redefinitions 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`:

```
1163 {*spider}
1164 \catcode'\_=\xwebCatLetter
1165 \catcode'\@=\xwebCatLetter
1166 /spider}
```

21.2 Some miscellaneous. . .

```
1167 {*spider}
1168 \let\amp\&      % ampersand
1169 \let\SS\S       % section sign
1170 \let\PP\P      % paragraph sign
1171 /spider}
```

21.3 The main section tag will be a mixture of `webfiles` and `webkernel.tex`:

#1 = number
#2 = (level and) title

```
1172 {*spider}
1173 \def\N#1.#2.{%
1174   \ifon\end{xwebModule}\fi    % webfiles
1175   \global\xwebGroupLevel 0% default value for group level
1176   \xweb_headcheck#2\xweb_headcheck    % spider: get group level and title
1177   {\let\*=\empty%
1178     \xdef\xweb_secno{#1}% webfiles: get the section number
1179   }%
1180   \message \expandafter{* \xweb_secno}%
1181   \xweb_PrepareSection{#1}%
1182   \ifon\begin{xwebModule}{\bf\xweb_ModTitle.}%
1183     % \xweb_ModTitle is generated by \xweb_headcheck.
1184     %\hskip 1em plus.1em minus.1em%
1185     \xweb_SpiderLopEntry{\xwebGroupLevel}%
1186 }
1187 /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.

```
1188 {*spider}
1189 \def\xweb_SpiderLopEntry#1{%
1190   \ifnum\c@\xwebLopDepth > \xwebGroupLevel
1191     \addcontentsline{lop}{starred}{%
1192       \protect\global\xwebGroupLevel #1 \thexwebModule.\~\xweb_ModTitle}%
1193     { \let\protect\@unexpandable@protect
1194       \edef\next{\write\xweb_cont{%
1195         \ZZ{\xweb_ModTitle}{\the\xwebGroupLevel}%
```

```

1196         {\xweb_secno}{\noexpand\thepage}}}%
1197     \next
1198     }% write "\ZZ{title}{depth}{sec}{page}" to .con file
1199     \fi
1200     \ifnum\c@xwebOutlineDepth > \xwebGroupLevel
1201         \xwebPDFOutline{\xweb_secno}{\xweb_ModTitle}% args: nr, title
1202     \fi}
1203 (/spider)

```

21.5 The following is from `webkernel`, with small additions for `webfiles`. This stuff is to allow initial =, 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

```

```

1204 (*spider)
1205 \newif\ifxweb_cancel\xweb_canceltrue
1206 \def\xweb_ifnextchar#1#2#3{\let\@tempe=#1\def\@tempa{#2}\def\@tempb{#3}%
1207     \xweb_ifnch}
1208 \def\xweb_ifnch{%
1209     \ifx \@tempc \@tempe\let\@tempd\@tempa
1210     \else\let\@tempd\@tempb\fi
1211     \@tempd}
1212 \def\xweb_makethechar#1{\let\@tempc=#1}
1213
1214 \def\xweb_headcheck#1#2\xweb_headcheck{%
1215     \xweb_makethechar{#1}%
1216     \def\theskipper{\vskip 3pt}%
1217     % extra skip before new starred module
1218     \def\xweb_ModTitle{#2}}%
1219     \xweb_ifnextchar={%                               % @*= title.
1220         \global\xwebGroupLevel 0% webfiles
1221         \ifnum \xwebGroupLevel<\c@xwebSecNoEject
1222             \def\theskipper{\xwebMainSecSkip}%
1223         \fi
1224         \xweb_canceltrue
1225     }{\xweb_ifnextchar1{%                               % @*1 title.
1226         \global\xwebGroupLevel 1% webfiles
1227         \xweb_cancelfalse
1228         \ifnum \xwebGroupLevel<\c@xwebSecNoEject
1229             \def\theskipper{\xwebMainSecSkip}%
1230         \fi
1231     }{\xweb_ifnextchar2{%                               % @*2 title.
1232         \global\xwebGroupLevel 2% webfiles
1233         \xweb_cancelfalse
1234     }{\xweb_ifnextchar3{%                               % @*3 title.
1235         \global\xwebGroupLevel 3% webfiles
1236         \xweb_cancelfalse
1237         \ifnum \xwebGroupLevel<\c@xwebSecNoEject
1238             \def\theskipper{\xwebMainSecSkip}%
1239         \fi
1240     }{\xweb_ifnextchar4{%                               % @*4 title.
1241         \global\xwebGroupLevel 4% webfiles
1242         \xweb_cancelfalse

```

```

1243 }{% else % @* title.
1244 \global\xwebGroupLevel 0% webfiles
1245 \ifxweb_cancel\else
1246 \ifnum \xwebGroupLevel<\c@xwebSecNoEject
1247 \def\theskipper{\xwebMainSecSkip}%
1248 \fi
1249 \fi
1250 \xweb_cancelfalse
1251 \def\xweb_ModTitle{#1{#2}}%
1252 }}}}}%
1253 \theskipper
1254 }
1255 \spider)

```

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

```

1256 (*spider)
1257 \def\CD#1\DC{#1}
1258 \spider)

```

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

```

1259 (*spider)
1260 \def\P{\B}
1261 %\let\P\B
1262 \spider)

```

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

```

1263 (*spider)
1264 \def\xweb_CweaveBindings{%
1265 \ifx \xweb_UserBindings\relax
1266 \xweb_rebind
1267 % indentation and paragraph layout
1268 \xweb_break \0% % SPIDER
1269 \xweb_IncrIndent \1%
1270 \xweb_DecrIndent \2%
1271 \xweb_ExprBreak \3%
1272 \xweb_backup \4%
1273 \xweb_OptBreak \5%
1274 \xweb_break \6%
1275 \xweb_BigBreak \7%
1276 \xweb_noindent \8%
1277 % C/C++ tokens
1278 \xwebRel \?%
1279 \xwebAddress \AND
1280 \xwebComplement \CM
1281 \xwebScope \DC
1282 \xwebEquiv \S % SPIDER
1283 \xwebGe \G
1284 \xwebRightShift \GG
1285 \xwebNe \I
1286 \xwebAssign \K
1287 \xwebLeftShift \LL
1288 \xwebMod \MOD
1289 \xwebNull \NULL
1290 \xwebNot \R

```

```

1291     \xwebBinOr           \OR
1292     \xwebMemberRef      \PA
1293     \xwebThis           \this
1294     \xwebOr             \V
1295     \xwebAnd            \W
1296     \xwebXor           \XOR
1297     \xwebLE            \L           % SPIDER
1298     \xwebPointer       \MG
1299     \xwebPointerMemberRef \MGA
1300     \xwebDecr          \MM
1301     \xwebIncr          \PP
1302     % more tokens
1303     \xwebId            \\\%
1304     \xwebIdLetter      \|\%
1305     \xwebRes           \&%
1306     \xwebString        \.\%       %% ( ...Emacs...
1307     \xwebStringBreak   \)\%
1308     \xwebNumber        \O           % SPIDER
1309     \xwebCombinedOp    \MRL
1310     % goes to TeX state
1311     \xwebComment       \C
1312     \xwebCxxComment    \SHC
1313     \xwebRefName       \X
1314     \xwebSpiderOutputFileName \XF
1315     % CWEB tokens
1316     \xwebMacrosHere    \ATH
1317     \xwebDefine        \D
1318     \xwebFormat        \F
1319     \xwebIdCat         \J
1320     \xwebVerbString    \=           % SPIDER
1321     % cross reference tags
1322     \xwebChangeFlag    \*%
1323     \xwebCRAlso       \A
1324     \xwebCRsAlso      \As
1325     \xwebCRCite       \Q
1326     \xwebCRsCite      \Qs
1327     \xwebCRUse        \U
1328     \xwebCRsUse       \Us
1329     \xwebCREt         \ET
1330     \xwebCRsEt        \ETs
1331     % finish the list
1332     \stop\stop
1333     \def\xweb_UserBindings{%
1334         \xweb_RestoreBindings
1335         \let\xweb_UserBindings\relax
1336     }%
1337     \fi
1338     }
1339 (/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.

```

1340 (*spider)
1341 \def\M#1.{%
1342     \ifon\end{xwebModule}\fi

```



```

1343 \xweb_PrepareSection{#1}%
1344 \ifon\begin{xwebModule}
1345   \ifnum\c@xwebLopDepth>9
1346     \addcontentsline{lop}{xwebsection}{%
1347       \protect\global\xwebGroupLevel 4 \thexwebModule.}%
1348     \edef\next{\write\xweb_cont{ZZ}{4}{\xweb_secno}%
1349       {\thepage}}\next % \ZZ{title}{depth}{sec}{page}%
1350     { \let\protect\@unexpandable@protect
1351     \edef\next{\write\xweb_cont{%
1352       \ZZ}{4}{\xweb_secno}{\thepage}}}%
1353   \next
1354   }% write "\ZZ{title}{depth}{sec}{page}" to .con file
1355   \fi
1356 }
1357 </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

1358 <*spider>
1359 \def\inx{
1360   \ifxwebIndex
1361     \xweb_SetupIndex
1362     \def\:{\I}
1363   \else
1364     \def\:#1.{} % gobble \:{{foo} 1, 2, 3.
1365   \fi
1366 }
1367 \def\fin{
1368   \ifxwebIndex
1369     \xweb_FinishIndex
1370   \fi
1371   \ifxwebRef
1372     \xweb_SetupReflist
1373     \def\:{\I}
1374   \else
1375     \def\:{ }
1376     \def\XF##1\XF{}
1377     \def\X##1\X{}
1378     \def\U##1.{ }
1379   \fi
1380 }
1381 \def\con{
1382   \ifxwebRef
1383     \endgroup
1384   \fi
1385 }
1386 </spider>

```

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

```

1387 <*spider>
1388 \def\xwebComment#1{%
1389     \5%           % 0.5em will be discarded on line break
1390     \hskip 1.5em
1391     \commentbegin%
1392     {\xweb_tex
1393     \xwebCommentHook
1394     #1%
1395     }%
1396     \commentend%
1397     }
1398 \let\xwebCommentHook\relax
1399 \let\xwebCxxComment\xwebComment
1400 </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.

```

1401 <*spider>
1402 \def\xwebDefine{\xweb_macro{define}}
1403 %\def\xwebFormat{\4\xweb_macro{format}}
1404 \def\xwebFormat\#1\#2\par{\ifxwebHideFormats\else
1405   \4\xweb_macro{format}\#{#1} \#{#2}\par\fi}
1406 </spider>

```

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

```

1407 <*spider>
1408 \def\xwebSpiderOutputFileName#1:#2\FX{\xwebRefName{#1}:\.#2\X}
1409 </spider>

```

21.15 Restore catcodes:

```

1410 <*spider>
1411 \catcode'\_=\xwebCatUsCode
1412 \catcode'\@=\xwebCatOther
1413 </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 L^AT_EX, the hyperref package*, October 1997.
- [5] Leslie Lamport. *MakeIndex: An Index Processor for L^AT_EX*, 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.

Symbols	<code>\arabic</code> 171, 237	<code>\ETs</code> <u>426</u> , <u>1244</u>
<code>\&</code> <u>426</u> , <u>1244</u>	<code>\As</code> <u>426</u> , <u>1244</u>	<code>\everypar</code> 253, 256
<code>\)</code> <u>426</u> , <u>1244</u>	<code>\ast</code> 705, 710	<code>\ExecuteOptions</code> 92
<code>*</code> <u>216</u> , <u>426</u> , <u>1244</u>	<code>\ATH</code> <u>426</u> , <u>1244</u>	<code>\exhyphenpenalty</code> 515, 523, 541
<code>\.</code> <u>426</u> , <u>1244</u>	<code>\ATL</code> <u>1137</u>	<code>\ext</code> 938, 940, 944, 950, 952, 956
<code>\?</code> <u>426</u> , <u>1244</u>		
<code>\@M</code> 398, 539, 541, 1072	B	F
<code>\@currentlabel</code> 251	<code>\B</code> <u>556</u> , 1241, 1242	<code>\F</code> <u>426</u> , <u>1244</u>
<code>\@defpar</code> 851, 857	<code>\bgroup</code> 250, 694, 1138	<code>\fbox</code> 726
<code>\@dottedtocline</code> 373, 383	<code>\Black</code> 202	<code>\fboxsep</code> 726
<code>\@flushglue</code> 1120		<code>\filbreak</code> 248
<code>\@for</code> 983	C	<code>\fin</code> 915, <u>1339</u>
<code>\@gobble</code> 331, 341, 357, 793	<code>\C</code> <u>426</u> , <u>1244</u>	<code>\first</code> 760, 763–768, 771, 774
<code>\@ifnextchar</code> 967	<code>\c@xwebLopDepth</code> 90, 91, 283, 303, 907, 1179, 1326	<code>\font</code> 618, 619
<code>\@input</code> 1084, 1133	<code>\c@xwebModule</code> 191, 745	<code>\footnotesize</code> 645, 740, 821, 1123
<code>\@mkboth</code> 319, 322	<code>\c@xwebOutlineDepth</code> 293, 1189	G
<code>\@noskipsecfalse</code> 255	<code>\c@xwebSecNoEject</code> 274	<code>\G</code> <u>426</u> , <u>1244</u>
<code>\@noskipsectrue</code> 258	<code>\CD</code> <u>1237</u>	<code>\ge</code> 609
<code>\@optfalse</code> 985	<code>\centerline</code> 910	<code>\GG</code> <u>426</u> , <u>1244</u>
<code>\@opttrue</code> 986–1000	<code>\ch</code> 917	<code>\gg</code> 650
<code>\@pnumwidth</code> 400	<code>\chapter</code> 318, 322	H
<code>\@secpenalty</code> 392, 416	<code>\chop</code> 940, 942, 952, 954	<code>\hangindent</code> 396, 560, 575, 598, 822, 1091, 1122
<code>\@starttoc</code> 320, 323	<code>\circ</code> 687	<code>\hfil</code> 581, 591, 870, 891
<code>\@tempa</code> 1195, 1198	<code>\clearpage</code> 266	<code>\hfill</code> 400
<code>\@tempb</code> 1195, 1199	<code>\CM</code> <u>426</u> , <u>1244</u>	<code>\hfilneg</code> 582, 592
<code>\@tempc</code> 1198, 1201	<code>\CodelineIndex</code> 12	<code>\hrule</code> 661
<code>\@tempd</code> 1198–1200	<code>\columnsep</code> 1049	<code>\hsize</code> 1071
<code>\@tempdima</code> 399	<code>\columnseprule</code> 1048	<code>\hskip</code> 261, 282, 558, 603, 704, 875–880, 1173, 1354
<code>\@tempe</code> 1195, 1198	<code>\commentbegin</code> 1355	<code>\hspace</code> 259
<code>\@tocrmarg</code> 397	<code>\commentend</code> 1360	<code>\hss</code> 400, 872, 892
<code>\@unexpandable@protect</code> 287, 306, 1182, 1331	<code>\con</code> 916, <u>1339</u>	<code>\hyperlink</code> 177
<code>\@</code> <u>426</u> , <u>1244</u>	<code>\cwebfile</code> <u>970</u>	<code>\hypertarget</code> 178
Numbers	D	<code>\hyphenpenalty</code> 514, 522, 540, 1072
<code>\0</code> <u>1244</u>	<code>\D</code> <u>426</u> , <u>1244</u>	I
<code>\1</code> <u>426</u> , <u>1244</u>	<code>\DC</code> <u>426</u> , 1238, <u>1244</u>	<code>\I</code> <u>426</u> , 1074, 1122, <u>1244</u> , 1342, 1347
<code>\2</code> <u>426</u> , <u>1244</u>	<code>\DeclareOption</code> 69–80, 84–86, 90, 91	<code>\if@noskipsec</code> 254, 419, 558, 562
<code>\3</code> <u>426</u> , <u>1244</u>	<code>\discretionary</code> 684	<code>\if@opt</code> 971, 1001
<code>\4</code> <u>426</u> , <u>1244</u>	<code>\divide</code> 417	<code>\ifcase</code> 873
<code>\5</code> <u>426</u> , <u>1244</u>	<code>\do</code> 983	<code>\iffalse</code> 220, 972
<code>\6</code> <u>426</u> , <u>1244</u>	<code>\DocInput</code> 29	<code>\IfFileExists</code> 884, 1042
<code>\7</code> <u>426</u> , <u>1244</u>	<code>\documentclass</code> 2, 59	<code>\ifmmode</code> 572, 730
<code>\8</code> <u>426</u> , <u>1244</u>		<code>\ifnum</code> 173, 176, 179, 192, 273, 274, 283, 293, 303, 360, 369, 770, 868, 887, 907, 1059, 1113, 1179, 1189, 1326
<code>\9</code> <u>1096</u>	E	<code>\ifon</code> <u>216</u> , 268, 280, 300, 302, 919, 1163, 1171, 1323, 1325
A	<code>\E</code> <u>426</u>	<code>\ifstr</code> <u>959</u> , 986–1000
<code>\A</code> <u>426</u> , <u>1244</u>	<code>\egroup</code> 262, 694, 1140	<code>\iftrue</code> 220, 973
<code>\addcontentsline</code> 284, 304, 908, 1180, 1327	<code>\EnableCrossrefs</code> 11	<code>\ifvmode</code> 561
<code>\addpenalty</code> 392	<code>\end</code> 32, 268, 300, 919, 1045, 1080, 1163, 1323	<code>\ifxweb_cancel</code> <u>1193</u>
<code>\addtoks</code> 756	<code>\endgraf</code> 573	<code>\ifxweb_FirstModule</code> <u>236</u>
<code>\addvspace</code> 393, 403	<code>\endinput</code> 1150	
<code>\aftergroup</code> 687, 688, 695	environments:	
<code>\amp</code> <u>1156</u>	<code>xwebModule</code> <u>24</u>	
<code>\anchor</code> 193, 194	<code>\equiv</code> 608	
<code>\AND</code> <u>426</u> , <u>1244</u>	<code>\ET</code> <u>426</u> , <u>1244</u>	

- `\ifxweb_GlobalCon` 68, 979
`\ifxweb_GlobalHideFormats` 67, 980
`\ifxweb_GlobalIndex` 63, 975
`\ifxweb_GlobalOC` 66, 978
`\ifxweb_GlobalRagged` 65, 977
`\ifxweb_GlobalRef` 64, 976
`\ifxweb_idxEntries` 842, 923
`\ifxweb_multicol` 1040, 1146
`\ifxweb_scnEntries` 842, 925
`\ifxwebCon` 62, 242
`\ifxwebHideFormats` 62, 807, 1368
`\ifxwebIndex` 62, 922
`\ifxwebRagged` 62, 247
`\ifxwebRef` 62, 924
`\ignorespaces` 577, 584
`\indent` 563
`\IndexPrologue` 6
`\input` 884, 918
`\interlinepenalty` 398
`\intro` 1032, 1033
`\inx` 914, 1339
`\it` 657, 687
- J**
- `\J` 426, 1244
- K**
- `\K` 426, 656, 1244
`\kern` 565, 575, 582, 622, 626–628, 632–634, 651, 657, 661, 662, 670, 687, 693, 740
- L**
- `\L` 1244
`\l@program` 327
`\l@starred` 353
`\l@xwebSection` 337
`\Lambda` 612
`\land` 607
`\langle` 734
`\large` 400
`\le` 610
`\leaders` 870
`\leavevmode` 410, 1092
`\listofprograms` 314
`\LL` 426, 1244
`\ll` 644
`\lnot` 613
`\lor` 614
`\lower` 622
- M**
- `\M` 298, 1321
`\makeatletter` 5
`\MakePrivateLetters` 1
`\maketoks` 748, 787
`\markboth` 7
`\mathbin` 645, 646, 648
`\mathchardef` 638
`\mathrel` 649, 656
`\maybe` 216, 972, 973
- `\mbox` 259
`\medskip` 1030, 1036
`\MessageBreak` 57, 1004, 1006
`\MG` 426, 1244
`\MGA` 426, 1244
`\mid` 640
`\MM` 426, 1244
`\MOD` 426, 1244
`\MRL` 426, 1244
- N**
- `\N` 265, 1161
`\name` 190, 194
`\ne` 611
`\newbox` 238, 548, 549, 621, 624, 630
`\newcommand` 322
`\newcounter` 88, 89, 213, 217
`\newdimen` 348, 546, 547
`\newenvironment` 240, 1044, 1047
`\newskip` 502, 503
`\next` 122, 126, 128, 150, 153, 155, 161, 165, 167, 288, 290, 307, 309, 361, 363, 365, 758, 771, 772, 777, 795–797, 799, 802, 851, 854, 857, 1183, 1186, 1329, 1330, 1332, 1334
`\noATL` 1137
`\noat1` 1137
`\nobreak` 400
`\noindent` 395, 565, 574, 822, 1029, 1035, 1111, 1138
`\nopagebreak` 1030, 1112
`\NULL` 426, 1244
`\null` 730
`\numberline` 331, 341, 357, 399
- O**
- `\O` 1244
`\onecolumn` 1052
`\onfalse` 928
`\OnlyDescription` 13
`\onmaybe` 216
`\ontrue` 219, 229
`\openin` 849
`\openout` 23, 244
`\oplus` 615
`\OR` 426, 1244
`\or` 874–879
- P**
- `\P` 1159, 1240
`\p@` 393, 403, 537, 726, 1049, 1068
`\PA` 426, 1244
`\PackageError` 54, 182, 1001
`\pagestyle` 4
`\parfillskip` 397, 1071, 1120
`\parindent` 259, 1069
`\parskip` 510, 518, 536, 1068
`\PB` 406, 794
`\pdfannotlink` 199
`\pdfdest` 205, 888, 1060, 1114
- `\pdfendlink` 203
`\pdfoutline` 194, 889, 1061, 1115
`\penalty` 247, 249, 256, 422, 581, 591
`\postATL` 1137
`\PP` 426, 1156, 1244
`\pretolerance` 513, 521, 539
`\PrintChanges` 31
`\PrintIndex` 30
`\ProcessOptions` 1145
`\protect` 58, 59, 285, 287, 305, 306, 1002, 1181, 1182, 1328, 1331
`\ProvidesPackage` 47
- Q**
- `\Q` 426, 1125, 1244
`\Qs` 426, 1126, 1244
`\quad` 1093, 1123
- R**
- `\R` 426, 1244
`\raise` 627, 633
`\rangle` 734
`\readcontents` 884, 894
`\RecordChanges` 12
`\Red` 202
`\renewcommand` 5
`\RequirePackage` 1147
`\reset@font` 740, 821, 1123
`\rightskip` 397, 511, 519, 537, 1070
`\rlap` 209
`\rm` 645, 694, 870
- S**
- `\S` 1158, 1244
`\sb` 694
`\scriptscriptstyle` 688
`\scriptstyle` 633
`\secstar` 222, 225
`\section` 6, 53, 319
`\setbox` 250, 332, 342, 358, 551, 552, 622, 625, 631, 787
`\setcounter` 88, 89, 214, 232
`\setdate` 21, 22
`\setname` 905, 906, 935, 943, 947, 955
`\setversion` 19, 20
`\sevensy` 617
`\sfcode` 512, 520, 538
`\SHC` 426, 1244
`\sim` 641
`\smallbreak` 868
`\smallskip` 423, 818
`\SS` 1156
`\st` 782
`\stop` 121, 143, 145, 149, 160, 493, 1313
`\string` 136, 163, 164
- T**
- `\T` 426

- \teni 617
 - \tentex 1140
 - \textbf 260, 281, 910
 - \texttt 1024, 1101
 - \textwidth 869, 891
 - \thepage 289, 308, 1185, 1330, 1333
 - \theskipper 1205,
1210, 1215, 1222, 1229, 1234
 - \thexwebModule 236,
286, 305, 746, 1181, 1328
 - \this 426, 1244
 - \tmpnr 189, 191, 744, 745
 - \tolerance 1072
 - \tt 400, 668, 684, 688
 - \tw@ 726, 810
 - \twocolumn 1050
 - \typeout 49, 984
- U**
- \U 426, 1127, 1244
 - \underline 1097
 - \unskip 1035
 - \unvbox 262
 - \uppercase 320, 323, 1139
 - \Us 426, 1128, 1244
 - \usepackage 3, 58
- V**
- \V 426, 1244
 - \vb 426
 - \vbox 250, 661
 - \vf 23, 24
 - \vfil 247
 - \vfilneg 247
 - \vskip 902, 910, 1029, 1109, 1205
- W**
- \W 426, 1244
 - \webfile 966, 1002, 1013
 - xwebModule 216
 - xwebSecNoEject 211
- X**
- \X 426, 720, 729, 1244, 1372
 - \XF 1244, 1372
 - \XOR 426, 1244
 - \xweb_AddTokens
... 755, 757, 773, 774, 779
 - \xweb_adn 757, 763–768
 - \xweb_anchor 170, 193
 - \xweb_backup 434, 595, 1253
 - \xweb_bak 545, 601
 - \xweb_bakk 545, 576, 583
 - \xweb_BigBreak .. 437, 570, 1256
 - \xweb_BoldTocLine ... 361, 381
 - \xweb_break 436, 570, 1249, 1255
 - \xweb_cancelfalse ... 1194,
1214, 1218, 1221, 1225, 1231
 - \xweb_canceltrue 1211
 - \xweb_CheckDot .. 734, 790, 1093
 - \xweb_consetup 864
 - \xweb_cont 244, 245, 288, 307,
864, 927, 1183, 1329, 1332
 - \xweb_contentsline ... 864, 893
 - \xweb_countA
... 753, 754, 758, 770, 781
 - \xweb_CR ... 95, 819, 1064, 1117
 - \xweb_CweaveBindings
... 102, 104, 109, 426, 1244
 - \xweb_decr 617, 642
 - \xweb_DocLayout 95, 526
 - \xweb_documentation ... 95, 233
 - \xweb_DoRebind 112
 - \xweb_ExprBreak . 433, 570, 1252
 - \xweb_file 968, 970
 - \xweb_FinishIndex . 1055, 1345
 - \xweb_FirstModulefalse 239, 243
 - \xweb_FirstModuletrue 912
 - \xweb_GlobalConfalse 77
 - \xweb_GlobalContrue 78
 - \xweb_GlobalHideFormatsfalse
..... 80
 - \xweb_GlobalHideFormatstrue 79
 - \xweb_GlobalHypertype
..... 83–86, 982
 - \xweb_GlobalIndexfalse ... 70
 - \xweb_GlobalIndextrue 69
 - \xweb_GlobalOCfalse 76
 - \xweb_GlobalOCtrue 75
 - \xweb_GlobalRaggedfalse .. 74
 - \xweb_GlobalRaggedtrue ... 73
 - \xweb_GlobalReffalse 72
 - \xweb_GlobalReftrue 71
 - \xweb_HasEntries . 846, 922, 924
 - \xweb_headcheck 1165, 1172, 1193
 - \xweb_hex 686
 - \xweb_hlink 170, 746
 - \xweb_htarget 170, 260
 - \xweb_hypertype 62,
173, 176, 179, 192, 887,
982, 998–1000, 1059, 1113
 - \xweb_ifnch 1193
 - \xweb_ifnextchar 1193
 - \xweb_incr 617, 643
 - \xweb_IncrIndent
..... 431, 595, 811, 1250
 - \xweb_indent 545, 560,
575, 597, 598, 600, 603, 810
 - \xweb_IndexTop 1026, 1058
 - \xweb_InputWebfile .. 901, 1012
 - \xweb_LayoutInit 529, 545
 - \xweb_macro 805, 1366, 1367, 1369
 - \xweb_makenote 770, 779
 - \xweb_makethechar 1193
 - \xweb_ModBox 236
 - \xweb_ModTitle . 1171, 1172,
1181, 1184, 1190, 1207, 1232
 - \xweb_multicolfalse 1041
 - \xweb_multicoltrue 1043
 - \xweb_noindent .. 438, 595, 1257
 - \xweb_NormalTocLine
..... 344, 363, 367
 - \xweb_oct 686
 - \xweb_off 970
 - \xweb_on 970
 - \xweb_opt .. 983, 984, 986–1000
 - \xweb_OptBreak .. 435, 570, 1254
 - \xweb_pdflink 180, 197
 - \xweb_pdftarget 181, 197
 - \xweb_pointer 617, 647
 - \xweb_poptoks 759, 762
 - \xweb_PrepareSection
... 216, 279, 301, 1170, 1324
 - \xweb_program 95, 559
 - \xweb_ProgramLayout
..... 105, 512, 526
 - \xweb_rebind 112, 429, 1247
 - \xweb_RestoreBinding . 145, 158
 - \xweb_RestoreBindings
..... 141, 495, 1315
 - \xweb_RestoreDocLayout 501, 532
 - \xweb_Rprogram 95, 409
 - \xweb_SaveBinding ... 124, 131
 - \xweb_SaveDocLayout .. 501, 530
 - \xweb_SaveExhyphenpenalty . 501
 - \xweb_SaveHyphenpenalty .. 501
 - \xweb_SaveParskip 501
 - \xweb_SavePretolerance ... 501
 - \xweb_SaveRightskip 501
 - \xweb_SaveSemSFCcode 501
 - \xweb_secno 216, 271,
273, 278, 289, 294, 1167,
1169, 1185, 1190, 1329, 1333
 - \xweb_SetupHrefs 170, 1011
 - \xweb_SetupIndex .. 1055, 1341
 - \xweb_SetupReflist . 1107, 1346
 - \xweb_SpiderLopEntry 1174, 1177
 - \xweb_star 216, 260
 - \xweb_StringSetup 666
 - \xweb_TagSec 780, 784
 - \xweb_tdone 771, 782
 - \xweb_tempa 961, 963
 - \xweb_tempb 962, 963
 - \xweb_tex 95,
706, 734, 1028, 1110, 1356
 - \xweb_TmpFile 846
 - \xweb_ToggleText
..... 730, 731, 733, 735
 - \xweb_toksA
749, 760, 762, 782, 787, 788
 - \xweb_toksB
750, 773, 774, 779, 782, 787
 - \xweb_toksC .. 751, 757, 780, 781
 - \xweb_toksD 752, 760, 773
 - \xweb_ToRestore 112, 137, 145, 146
 - \xweb_TTLopLine 334, 390
 - \xweb_undefine 141
 - \xweb_undefined 112, 134, 143, 144
 - \xweb_UserBindings
..... 95, 426, 930, 1244
 - \xwebAddress 441, 637, 1260
 - \xwebAnd 457, 606, 1276
 - \xwebAssign 448, 637, 1267
 - \xwebBinOr 453, 637, 1272
 - \xwebCatActive 34

- \xwebCatClose [34](#)
 \xwebCatEscape [34](#)
 \xwebCatIgnore [34](#)
 \xwebCatLetter . . . [34](#), 1153, 1154
 \xwebCatOpen [34](#)
 \xwebCatOther [34](#), 1376
 \xwebCatUsCode . . . [34](#), 1149, 1375
 \xwebChangeFlag
 483, [837](#), 1129, 1303
 \xwebCombinedOp . . . 471, [655](#), 1290
 \xwebComment 473, [701](#), 1292, [1351](#)
 \xwebCommentHook . . . [701](#), [1351](#)
 \xwebComplement . . . 442, [637](#), 1261
 \xwebConfalse 979, 994
 \xwebContents 242, [882](#)
 \xwebContentsBot 895, [901](#)
 \xwebContentsTop 886, [901](#)
 \xwebContrue 979, 995
 \xwebCRAlso 484, [827](#), 1304
 \xwebCRChanged 917, [1015](#)
 \xwebCRCite 486, [827](#), 1306
 \xwebCREt 490, [837](#), 1310
 \xwebCrossRef [815](#),
 828, 829, 831, 832, 834,
 835, 1017, 1102–1105, 1123
 \xwebCRsAlso 485, [827](#), 1305
 \xwebCRsCite 487, [827](#), 1307
 \xwebCRsEt 491, [837](#), 1311
 \xwebCRsUse 489, [827](#), 1309
 \xwebCRUse 488, [827](#), 1308
 \xwebCxxComment
 474, [701](#), 1293, [1351](#)
 \xwebDate [15](#), 49
 \xwebDecr 462, [637](#), 1281
 \xwebDecrIndent . . . 432, [595](#), 1251
 \xwebDefine 478, [805](#), 1298, [1365](#)
 \xwebEquiv 444, [606](#), 1263
 \xwebFormat 479, [805](#), 1299, [1365](#)
 \xwebGe 445, [606](#), 1264
 \xwebGroupLevel
 [211](#), 269, 274, 285, 305,
 329, 339, 355, 360, 369,
 370, 372, 374, 376, 385,
 1164, 1174, 1179, 1181,
 1184, 1189, 1209, 1213,
 1217, 1220, 1224, 1227, 1328
 \xwebHideFormatsfalse 981, 997
 \xwebHideFormatstrue . 980, 996
 \xwebId 465, [655](#), 1284
 \xwebIdCat 480, [715](#), 1300
 \xwebIdIndex 923, 944, [1055](#)
 \xwebIdLetter 466, [655](#), 1285
 \xwebIncr 463, [637](#), 1282
 \xwebIndentUnit 545, 560,
 565, 582, 597, 600, 603, 810
 \xwebIndexDeclared . . 1075, [1096](#)
 \xwebIndexEntry 1074, [1088](#)
 \xwebIndexfalse 975, 987
 \xwebIndexIntro [1021](#), 1032, 1035
 \xwebIndexName [1021](#), 1030
 \xwebIndextrue 975, 986
 \xwebInputIndex [933](#)
 \xwebInputReflist [933](#)
 \xwebJobName [1055](#)
 \xwebJobname 171,
 242, 244, 315, 849, 852,
 905, 908, 910, 923, 925,
 936, 944, 948, 956, 1024,
 1060, 1061, [1100](#), 1114, 1115
 \xwebLapStar [208](#), 1076
 \xwebLE 459, [606](#), 1278
 \xwebLeftShift 449, [637](#), 1268
 \xwebLopIndent [347](#),
 376, 377, 385, 386, 396, 399
 \xwebLopIndentMaxLevel
 [347](#), 369, 370
 \xwebLopName [314](#)
 \xwebMacrosHere 477, [715](#), 1297
 \xwebMainSecSkip
 [265](#), 1210, 1215, 1222, 1229
 \xwebMajor [15](#)
 \xwebMajorNr 18, 19
 \xwebMemberRef 454, [637](#), 1273
 \xwebMod 450, [637](#), 1269
 xwebModule (environment) . . . [236](#)
 \xwebNe 447, [606](#), 1266
 \xwebNot 452, [606](#), 1271
 \xwebNull 451, [606](#), 1270
 \xwebNumber 470, [686](#), 1289
 \xwebNumberListHangindent
 [815](#), 1091, 1122
 \xwebOr 456, [606](#), 1275
 \xwebPageName [882](#)
 \xwebPDFOutline [187](#), 294, 1190
 \xwebPointer 460, [637](#), 1279
 \xwebPointerMemberRef
 461, [637](#), 1280
 \xwebProgPenalty [414](#)
 \xwebRaggedfalse 977, 991
 \xwebRaggedtrue 977, 990
 \xwebReffalse 976, 989
 \xwebReflist 925, 956, [1107](#)
 \xwebReflistName [1100](#), 1112
 \xwebRefMacrosHere [715](#)
 \xwebRefName
 475, 720, [728](#), 1294, 1372
 \xwebRefNumber 719, 734, [738](#)
 \xwebReftrue 976, 988
 \xwebRel 440, [637](#), 1259
 \xwebRes 467, 653, [655](#), 812, 1286
 \xwebRightShift 446, [637](#), 1265
 \xwebRLCite [1100](#), 1125
 \xwebRLsCite [1100](#), 1126
 \xwebRLsUse [1100](#), 1128
 \xwebRLUse [1100](#), 1127
 \xwebScope 443, [637](#), 1262
 \xwebSectionName [882](#)
 \xwebSetModNrList
 741, [784](#), 823, 1093, 1124
 \xwebSetModuleNumber
 [743](#), 785, 871
 \xwebSpiderOutputFileName
 1295, 1372
 \xwebSpiderOutputFilename [1371](#)
 \xwebString
 468, [666](#), 716, 726, 797, 1287
 \xwebStringBreak 469, [666](#), 1288
 \xwebThis 455, [637](#), 1274
 \xwebTocName [314](#)
 \xwebVerbString 481, [725](#), 1301
 \xwebVersion [15](#), 49
 \xwebXor 458, [606](#), 1277

 Y
 \Y [414](#), 587, 818

 Z
 \Z [426](#)
 \z@skip 536
 \ZZ 289, 291,
 308, 310, [864](#), 893, 1184,
 1187, 1329, 1330, 1333, 1335