The \texttt{webfiles} Package

Version 0.5.39

Mark Potse *

1999/05/26

1 Introduction

The \texttt{webfiles} package makes it possible to plug the documentation of \texttt{CWEB} and Spidery \texttt{WEB} programs into a \LaTeX\ document. One can include any number of webs by including the \texttt{weave} processor’s output with the command

\begin{verbatim}
\webfile[(options)](filename)
\end{verbatim}

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

\begin{verbatim}
\usepackage[(options)]{webfiles}
\end{verbatim}

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

This manual does not describe the usage of \texttt{WEB} systems themselves; they have their own manuals\cite{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 \texttt{cweb} style by Joachim Schrod (even this manual is based on his). Apart from the fact that, with this package, webs 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 \texttt{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 \texttt{WEB} to see what ‘on’ means in this context.)
8. The \texttt{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 \texttt{multicol} cannot be found, \LaTeX's \texttt{twocolumn} is used.
9. The index and list of refinements can be suppressed in the \LaTeX\ document.
10. Format (\texttt{\@f}) statements can be suppressed in the \LaTeX\ document.
11. Support for hyperlinks is included, as well as direct support for \texttt{PDF\LaTeX}.

\footnote{Medical Physics Department, University of Amsterdam.}
\footnote{Email: M.Potse@amc.uva.nl}
\footnote{version 3, this package does not work with older versions}
1.1 The overall structure

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

\texttt{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 \texttt{CWEB} sources; I refer henceforth to it as the “plain version.”)

The document structure is tagged in the mother document, with \texttt{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:

\begin{verbatim}
\usepackage{webfiles}
\end{verbatim}

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

\begin{verbatim}
\webfile{(filename)}
\end{verbatim}

where \texttt{(filename)} is the .tex file that is output by a \texttt{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 \texttt{webfile} command and the \texttt{usepackage(webfiles)} command accept an optional argument that may contain a comma-separated list of options. Options specified to \texttt{usepackage} act as global defaults that can be overridden in the optional argument to \texttt{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 \texttt{weave} with the \texttt{-x} option\footnote{The \texttt{-x} option suppresses output of the index and refist—see the documentation.}. You can specify an introductory text for the index with the tag \texttt{xwebIndexIntro}, the introduction is the argument of this tag. To suppress generation of an identifier index, even when \texttt{weave} was not called with the \texttt{-x} option you can use an optional argument to the \texttt{webfile} command; for example:

\begin{verbatim}
\webfile[noindex]{foo}
\end{verbatim}

To suppress identifier indexes in all web’s, use this option in the \texttt{usepackage} command:

\begin{verbatim}
\usepackage[noindex]{webfiles}
\end{verbatim}

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

The same applies to the list of refinements, using the options \texttt{reflist} and \texttt{noreflist}.
2.2 The table of contents

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

The tables of contents are produced like all \LaTeX{}'s tables of contents etc., in the second \LaTeX{} run. They are stored in a file with name \langle basename\rangle.con, where \langle 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 \LaTeX{} provides. The list can be produced with a

\listofprograms

command, which can be placed at any position. It is customary to put it at the beginning of the mother document, after it’s table of contents. As with the List of Figures, the List of Programs is generated at the second \LaTeX{} run. It can list programs, main sections and normal sections; the default is to list only programs and main sections. You can control what goes into the list by setting the counter \xwebLopDepth:\(^3\)

\(^3\)I consider releasing a revised version of Spider that is more compatible with CWEB, and has less bugs
<table>
<thead>
<tr>
<th>option</th>
<th>opposite</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>index*</td>
<td>noindex</td>
<td>print identifier index</td>
</tr>
<tr>
<td>noindex</td>
<td>index</td>
<td>don't print identifier index</td>
</tr>
<tr>
<td>reflist*</td>
<td>noreflist</td>
<td>print List of Refinements</td>
</tr>
<tr>
<td>noreflist</td>
<td>reflist</td>
<td>don't print List of Refinements</td>
</tr>
<tr>
<td>contents*</td>
<td>nocon</td>
<td>print table of contents</td>
</tr>
<tr>
<td>nocon</td>
<td>contents</td>
<td>don't print table of contents</td>
</tr>
<tr>
<td>allsections*</td>
<td>onlychanges</td>
<td>print all sections</td>
</tr>
<tr>
<td>onlychanges</td>
<td>allsections</td>
<td>print only changed sections</td>
</tr>
<tr>
<td>showformats*</td>
<td>hideformats</td>
<td>show @f declarations</td>
</tr>
<tr>
<td>hideformats</td>
<td>showformats</td>
<td>don't show @f declarations</td>
</tr>
<tr>
<td>raggedbottom*</td>
<td>flushbottom</td>
<td>try not to break modules across pages</td>
</tr>
<tr>
<td>flushbottom</td>
<td>raggedbottom</td>
<td>try to create a flush bottom, break modules if necessary</td>
</tr>
<tr>
<td>hyperref</td>
<td>nohype</td>
<td>make hyperlinks using hyperref package</td>
</tr>
<tr>
<td>pdfex</td>
<td>nohype</td>
<td>make hyperlinks and PDF outlines with PDFTeX</td>
</tr>
<tr>
<td>nohype*</td>
<td>...</td>
<td>no hyperlinks</td>
</tr>
<tr>
<td>nofiles</td>
<td>writefiles</td>
<td>don't write a contents file (equivalent to setting \setcounter{xwebLopDepth}{-9}).</td>
</tr>
<tr>
<td>writefiles*</td>
<td>nofiles</td>
<td>do write a contents file. (equivalent to setting \setcounter{xwebLopDepth}{9}).</td>
</tr>
</tbody>
</table>

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 in between \webfile commands you can handle each WEB file differently, if you wish. The default value is 9. It is a \LaTeX{} counter, so its value can be changed using the \setcounter and \addtocounter commands.

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

The entries for the List of Programs are put into the file “\jobname.lop”, where \jobname is the jobname that \LaTeX{} chose for the mother document (usually the name of the .tex file, without the extension). Edit this file only if you’re desperate, 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[5]. To use this feature, you have to import the hyperref package yourself, before importing the webfiles package, e.g.:

\usepackage{hypertext}{hyperref}
Alternatively, with the `pdftex` option, direct support for PDFLaTeX can be enabled. This provides the same look and feel as the `pdfcwebmac` macros, including the ‘PDF Outline’, but of course works only with PDFLaTeX.

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.

```latex
\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\xwebCRChanged{%
  \xwebCrossRef{The following sections were changed by the change file:}%
  \let\maybe=\iffalse}
\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 `\arabic{xwebModule}`. Its default definition is `\arabic{xwebModule}`.

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

## 3 Problems and Restrictions

Restrictions:

- Please be aware that the vertical bar (`|`) is used by WEB to delimit small program code pieces in the documentation parts, and is therefore processed by `weave`. You cannot use it for `\LaTeX` anymore.
In particular, you cannot specify rules for the `tabular` or the `array` environment. Since you probably want to do so, you have two choices left:

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

   \begin{tabular}{|l|l|}
   you have to write

   \begin{tabular}{|I|I|}

2. Use ‘\^c’ instead of ‘|’. I.e., instead of \begin{tabular}{|l|l|} you may write \begin{tabular}{|1\^c|1}. 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 ‘\@.\.,\@’, ‘\@\.,\@’, or even ‘\@\.,{foolish}@’. Of course you may write ‘\@\.,.accent@’ or ‘\@< Handle accent \.,@’.

   • 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:

\begin{verbatim}
\ATL
\B
\M
\N
\PB
\Y
\end{verbatim}

\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 \@{\texttt{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 \@1 redefinitions is not proper. But it wasn’t in the plain version, either.

Problems:
• One cannot use an other basic font size than 10 pt. A few symbol definitions and layout parameters depend on this.

• C++ comments in CWEB (i.e., from // to the end of the line) are typeset as C comments. This is especially bad if they are used for a whole block of comment lines, as it is quite common. Please put such comment blocks in the documentation part.

4 \TeXX nical Data

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

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

There are now two kinds of indexes, both optional: Those that are created by \texttt{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 \texttt{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 \texttt{xwebModule}. They are not made into \texttt{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 \texttt{weave} will always be right. (It refers literally to the section numbers.)

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

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

This package reserves the namespace \texttt{xweb}.

4.1 Files

\texttt{webfiles.sty} is the main package file. It suffices to handle CWEB files.

\texttt{cwebmac.tex} is the macro package that is input on the first line of the \texttt{.tex} file that \texttt{weave} creates. It contains macros that enable \TeX to handle CWEB, but it can see if it’s being input by \LaTeX (instead of \TeX), and, if so, it won’t define anything.

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

\texttt{webkernel.tex} is input by \texttt{xweb.tex}; it is the Spidery analogon of \texttt{cwebmac.tex}, but it contains no language-dependent macros: These are concentrated in the \texttt{xweb.tex} files. If \texttt{webkernel.tex} sees that it is being input by \LaTeX, it inputs \texttt{swebbind.sty}, and doesn’t define anything, else it defines the macros that enable \TeX to handle a Spidery WEB.

\texttt{swebbind.sty} contains redefinitions of some macros in \texttt{webfiles.sty} that make them able to handle Spidery WEBs.

\texttt{webfiles.sty} and \texttt{swebbind.sty} are supplied with the \texttt{webfiles} package. So is \texttt{webkernel.tex}; this file replaces the file of the same name that goes together with Spider. The \texttt{xweb.tex} files are supplied with their respective (Spidery) WEBs. \texttt{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 \LaTeXe. \\

\begin{verbatim}
\documentclass[twoside]{ltxdoc}
\usepackage{xwebdoc,array,cwebarray}
\pagestyle{headings}
\renewcommand{\MakePrivateLetters}{\makeatletter_/\relax}
\IndexPrologue{\section*(Index)%
\markboth{Index}{Index}%

The italic numbers denote the code lines where the corresponding
entry is described, underlined numbers point to the definition,
all others indicate the places where it is used.}
\EnableCrossrefs
\CodelineIndex \RecordChanges
\OnlyDescription
\</driver\}
\end{verbatim}

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.

\begin{verbatim}
\documentclass[twoside]{ltxdoc}
\usepackage{xwebdoc,array,cwebarray}
\pagestyle{headings}
\renewcommand{\MakePrivateLetters}{\makeatletter_/\relax}
\IndexPrologue{\section*(Index)%
\markboth{Index}{Index}%

The italic numbers denote the code lines where the corresponding
entry is described, underlined numbers point to the definition,
all others indicate the places where it is used.}
\EnableCrossrefs
\CodelineIndex \RecordChanges
\OnlyDescription
\</driver\}
\end{verbatim}
5.2 After that, the document can be input by the driver.

32 \texttt{\textbackslash driver}
33 \texttt{\begin{document}}
34 \texttt{\textbackslash DocInput\{webfiles.dtx\}}
35 \texttt{\textbackslash PrintIndex}
36 \texttt{\textbackslash PrintChanges}
37 \texttt{\textbackslash end\{document\}}
38 \texttt{\textbackslash /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 CWB 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 CWB file: The document structure and the program structure.

A CWB 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 CWB 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 CWB 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 \webCatUsCode. This method is better than to use a group, not all cseqs must be defined global this way.

6.2 Let's identify this package against the user and in the Log file.
The very first (alpha) version of the `cweb` style was a style option. The version on which the `webfiles` style is based was a full style. `webfiles.sty` itself is an option again, and since the author has LaTeX 2ε, it has become a `package`. Therefore we test if a documentclass was chosen (by testing if `\section` is defined).

\begin{verbatim}
57  \ifx \section\undefined
58    \PackageError{webfiles}{webfiles is not a document class, but only a package.}
59    \immediate \write18{\PackageError{webfiles}{webfiles is not a document class, but only a package.}
50            Please adapt your documentclass tag appropriately.
51            I.e., write \MessageBreak
52            \protect\usepackage{webfiles} instead of
53            \protect\documentclass{webfiles}.
54    \fi
55  \end{verbatim}

66  \end{verbatim}
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.

\newif\ifxweb_GlobalIndex \newif\ifxwebIndex
\newif\ifxweb_GlobalRef \newif\ifxwebRef
\newif\ifxweb_GlobalRagged \newif\ifxwebRagged
\newif\ifxweb_GlobalOC
\newif\ifxweb_GlobalHideFormats \newif\ifxwebHideFormats
\newif\ifxweb_GlobalCon \newif\ifxwebCon
\DeclareOption{index}{\xweb_GlobalIndextrue}
\DeclareOption{noindex}{\xweb_GlobalIndexfalse}
\DeclareOption{reflist}{\xweb_GlobalReftrue}
\DeclareOption{noreflist}{\xweb_GlobalReffalse}
\DeclareOption{raggedbottom}{\xweb_GlobalRaggedtrue}
\DeclareOption{flushbottom}{\xweb_GlobalRaggedfalse}
\DeclareOption{onlychanges}{\xweb_GlobalOCtrue}
\DeclareOption{allsections}{\xweb_GlobalOCfalse}
\DeclareOption{nocon}{\xweb_GlobalConfalse}
\DeclareOption{contents}{\xweb_GlobalContrue}
\DeclareOption{hideformats}{\xweb_GlobalHideFormastrue}
\DeclareOption{showformats}{\xweb_GlobalHideFormatsfalse}
\newcount\xweb_hypertype
\newcount\xweb_GlobalHypertype
\DeclareOption{hyperref}{\xweb_GlobalHypertype=1}
\DeclareOption{pdftex}{\xweb_GlobalHypertype=2}
\DeclareOption{nohype}{\xweb_GlobalHypertype=0}
\newcounter{xwebLopDepth} \setcounter{xwebLopDepth}{9}
\newcounter{xwebOutlineDepth} \setcounter{xwebOutlineDepth}{9}
\DeclareOption{nofiles}{\x@xwebLopDepth=-9}
\DeclareOption{writefiles}{\x@xwebLopDepth=9}
\ExecuteOptions{index,reflist,raggedbottom,allsections,}
\ExecuteOptions{contents,showformats,nohype,writefiles}

\end{main}
Chapter 8: The Interface between cweave and TeX.

Here we present all tags output by cweave in an ordered fashion. First we look at those tags which are part of the 'protected interface,' i.e., 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.

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.

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

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

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

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

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

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

\footnote{This information refers to CWEB 3.0.}
8.3 Some tags appear only in special circumstances and may therefore be considered as private tags. The largest part of them concern the tagging of program code, we’ll have a look at them later. First we present the tags used in other areas.

Lists of section numbers occur 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 \&, listed above, appears also in the index.

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

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

\I start of a new refinement name

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

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

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

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

CWEB itself introduces its own operators:

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

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

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

Program code must be indented according to its structure. Each “larger” statement is typeset as one paragraph, usually only one line long. The basic indentation of the next statement may be incremented and decremented in given units. If a statement has to be broken in more than one line nevertheless, a hanging indentation is added to the basic indentation for the subsequent lines. One can mark an ‘optional’ statement start, ie, some kind of optional paragraph start. If a line break must be inserted it should be inserted here and no hanging indentation should be added to the basic indentation.
\1 future stmts indented one more unit
\2 future stmts indented one less unit
\3 \1 optional line break within a statement
   Arg. 1: digit, penalty for line break
\4 backspace one indentation unit
\5 optional line break or small space (between run-in statements)
\6 line break
\7 line break and additional vertical space
\8 line is a preprocessor directive (must be issued at start of line)

8.5 Some args are designated as strings above. Within these args the following tags are used to represent special characters:
\_ space
\& ampersand
\backslash backslash
\^ hat
\_ underscore
\{ left brace
\} right brace
\~ tilda

8.6 Last, we have tags which are used in their “standard” \LaTeX meaning. We don’t have much work with them:
\# hash mark in program or in string
\$ dollar
\% percent
\lbrace block start (is set in math mode)
\rbrace block end (is set in math mode)
\, thin space (like the \LaTeX default)
\langle left delimiter in include directives, or in templates
\le C operator: less or equal
\cdots Function prototypes: Variable number of arguments
\rangle right delimiter in include directives, or in templates
9 Processing states.

We have to typeset five different categories of material: Documentation, program pieces—embedded within the documentation and as large chunks, $\text{T}_{\text{E}}X$ 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. A \texttt{\textbackslash \textbackslash B} switches to program state. This can happen in documentation and program state.

3. While we process the argument of \texttt{\textbackslash PB} we’re in restricted program state; \texttt{\textbackslash PB} may appear in documentation and in $\text{T}_{\text{E}}X$ state. Since $\text{T}_{\text{E}}X$ state can be switched on within restricted program state, \texttt{\textbackslash PB} can appear within the argument of itself.

4. In the arguments of \texttt{\textbackslash C}, \texttt{\textbackslash SHC}, and \texttt{\textbackslash X} we switch to $\text{T}_{\text{E}}X$ 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.

\texttt{\textbackslash xweb-documentation} will switch to documentation state, \texttt{\textbackslash xweb-program} to program state, \texttt{\textbackslash xweb-Rprogram} to restricted program state, \texttt{\textbackslash xweb_{text}} to $\text{T}_{\text{E}}X$ state, and \texttt{\textbackslash 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 \texttt{\textbackslash cweave}, and tagged by \texttt{\textbackslash cweave}. The used tags are from a global namespace and should only be in effect during these states. We call this tag set the \texttt{\textbackslash CWEAVE} bindings. In the other two states the tags are largely defined by the user, the tag set is called the \texttt{\textbackslash 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 restauraion of the user binding; it will be done automatically by $\text{T}_{\text{E}}X$ at the end of the group. Nevertheless we must be able to switch from the \texttt{\textbackslash cweave} bindings back to the user bindings which were in effect when we activated the \texttt{\textbackslash cweave} bindings. This is needed for the $\text{T}_{\text{E}}X$ 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 $\text{T}_{\text{E}}X$ 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.

<table>
<thead>
<tr>
<th>STATE</th>
<th>BINDING</th>
<th>LAYOUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>documentation</td>
<td>U</td>
<td>D</td>
</tr>
<tr>
<td>program</td>
<td>C</td>
<td>P</td>
</tr>
<tr>
<td>restricted program</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>TeX</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

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 \web_CweaveBindings and \web_ProgramLayout, we redefine \web_UserBindings and \web_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.

```latex
\begin{verbatim}
100 \def\web_UserBindings{\relax}
101 \def\web_ProgramLayout{\relax}
102 \def\web_CweaveBindings{}
103 \def\web_DocLayout{}
104 \let\web_DocLayout=\relax
105 \let\web_UserBindings=\relax
106 \def\web_CweaveBindings{}
107 \def\web_ProgramLayout{}
108 \def\web_CweaveBindings{}
109 \def\web_ProgramLayout{}
110 \let\web_CweaveBindings=\relax
111 \def\web_DocLayout{}
112 \let\web_DocLayout=\relax
113 \def\web_CweaveBindings{}
114 \def\web_DocLayout{}
115 \let\web_DocLayout=\relax
116 \let\web_DocLayout=\relax
117 \let\web_DocLayout=\relax
\end{verbatim}
```
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 \texttt{\textbackslash{}foo}, i.e., of the cseq with the name \texttt{\textbackslash{}foo}, is saved as the cseq with the name \texttt{\textbackslash{}xweb\_s\_\textbackslash{}\textbackslash{}string\textbackslash{}foo\endcsname}, i.e., as \texttt{\textbackslash{}csname xweb\_s\_\textbackslash{}\textbackslash{}string\textbackslash{}foo\endcsname}.

10.1 The save process is not done statically, but by the macro \texttt{\textbackslash{}xweb\_rebind} which interprets a list of tuples \texttt{(old\_name,new\_name)}, terminated by the tuple \texttt{(#stop,stop)}. Eventually it constructs two new lists, \texttt{\textbackslash{}xweb\_To\_Restore} with the cseqs which had a binding, and \texttt{\textbackslash{}xweb\_undefined} with the names which didn’t have one.

The saving is actually done by \texttt{\textbackslash{}xweb\_Save\_Binding}, \texttt{\textbackslash{}xweb\_rebind} is responsible for the initialization, \texttt{\textbackslash{}xweb\_Do\_Rebind} for the effective rebounding and the tail recursion on the list.

\texttt{\textbackslash{}def \textbackslash{}xweb\_rebind}{%}
\begin{verbatim}
\xweb\_undefined\{%
\xweb\_To\_Restore\{%
\xweb\_Do\_Rebind
\}
\def \textbackslash{}xweb\_Do\_Rebind\#1\{%
\ifx \#1\stop
\let \next\relax
\else
\xweb\_Save\_Binding \#2\%
\let \#2=\#1%
\let \next\xweb\_Do\_Rebind
\fi
\next
\}
\end{verbatim}
\texttt{\textbackslash{}def \textbackslash{}xweb\_Do\_Rebind\#1\{%}
\ifx \#1\stop
\else
\xweb\_Save\_Binding \#2\%
\let \#2=\#1%
\let \next\xweb\_Do\_Rebind
\fi
\next
\}
\end{verbatim}

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{}ical note: The cseq-name for the saved binding must be created before the \texttt{\textbackslash{}let} is executed.

\begin{verbatim}
\def \xweb\_Save\_Binding\#1{%
\ifx \#1\undefined
\xweb\_undefined \expandafter{\the\xweb\_undefined \#1}%
\else
\expandafter{\let \csname xweb\_s\_\textbackslash{}\textbackslash{}string\#1\endcsname =\#1}
\end{verbatim}

\texttt{\textbackslash{}def \textbackslash{}xweb\_Save\_Binding\#1{%}
\ifx \#1\undefined
\else
\expandafter{\let \csname xweb\_s\_\textbackslash{}\textbackslash{}string\#1\endcsname =\#1}
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 \web_undefine and \web_RestoreBinding iterate over a list of cseqs terminated by \stop.

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

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

Another \next assignment.
11 Hyperlinks.

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

The anchor name depends on the current value of the \texttt{xwebModule} counter; this value must be set before \texttt{xweb\_hyperlink} or \texttt{xweb\_hypertarget} is used.

\begin{verbatim}
175 \def\xweb\_anchor\{xweb\texttt{mod}:\xweb\texttt{Jobname}:\texttt{arabic}\{xweb\texttt{Module}\}\}
176 \def\xweb\_SetupHrefs\{  
178  \ifnum\xweb\_hypertype=0  
179     \def\xweb\_hlink\{\#\#1\{\#\#1\}
180     \def\xweb\_htarget\{\#\#1\}
181  \else\ifnum\xweb\_hypertype=1  
182     \def\xweb\_hlink\{\hyperlink{\xweb\_anchor}{\#\#1}\}
183     \def\xweb\_htarget\{\hypertarget{\xweb\_anchor}{\#\#1}\}
184  \else\ifnum\xweb\_hypertype=2  
185     \def\xweb\_hlink\{\xweb\_pdflink{\xweb\_anchor}{\#\#1}\}
186     \def\xweb\_htarget\{\xweb\_pdtarget{\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 \end{verbatim}
12 PDFTeX support.

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

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

192 \def\xwebPDFOutline#1#2{% 
193 \edef\tmpnr{#1} 
194 \edef\name{#2} 
195 \ifnum\xweb\hypertype=2
196 \edef\anchor{\xweb_anchor} 
197 \pdfoutline goto name{\anchor}{#1 \name} 
198 \fi 
199 } 
200 \pdfendlink 
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 \pdfendlink 
206 \edef\xweb_pdftarget#1#2{% 
207 \pdfdest name{#1} fitbh 
208 #2 
209 } 
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.

\begin{verbatim}
213 \(*main\)
214 \def\xwebLapStar{\rlap{/*/}}
215 \end{verbatim}

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.

\begin{verbatim}
216 \(*main\)
217 \newcount\xwebGroupLevel
218 \newcounter{\xwebSecNoEject}
219 \setcounter{\xwebSecNoEject}{3}
220 \end{verbatim}

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 \texttt{\xwebModule}. The number can contain a star, to indicate that the section is changed by a changefile. In that case, \texttt{\ifon} is made \texttt{true}; else it will be \texttt{\maybe}, and by redefining \texttt{\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 \texttt{\xweb\_PrepareSection}, which is used before each section. This macro also changes to documentation state.

\begin{verbatim}
221 \(*main\)
222 \newcounter{\xwebModule}
223 \newif\ifon \% determines if the section is ‘on’
224 \def\onmaybe{\let\ifon=\maybe} \% analogous to \texttt{\ontrue}
225 \let\maybe=\iftrue \% initial meaning; the user can say "\texttt{\let\maybe=\iffalse}"
226 \def\xweb\_PrepareSection\#1\%
227 {\let\secstar=\#1}\%
228 \let\*=\empty \% remove * from the number
229 \def\xweb\_secno\secstar \% if they are the same; i.e.:
230 \ifx\xweb\_secno\secstar \% if there was no star in \#1
231 \onmaybe \% if there was no star in \#1
232 \else
233 \ontrue
234 \fi
235 \def\xweb\_star\*\%
236 \fi
237 \setcounter{\xwebModule}{\#1}\% global set!
238 \xweb\_documentation
239 \}
240 \end{verbatim}
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 unvboxed, but this is more difficult. Must think about it.

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

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

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

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

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

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

When writing to the \texttt{.con} file, we have to use \texttt{@unexpandable@protect}; else there will be trouble with insertion of \texttt{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 \texttt{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).

\begin{verbatim}
272 \{main\}
273 \def\xwebMainSecSkip{\clearpage}
274 \def\xwebSecNoEject{3,}%
275 \ifon\end{xwebModule}\fi
276 \global\xwebGroupLevel #1 % <-- space!!
277 \let\*=\empty%
278 \edef\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 em plus.1em minus.1em%
291 \ifnumc@xwebLopDepth > #1%
292 \addcontentsline{lop}{starred}{% space after #1 !
293 \let\protect\global\xwebGroupLevel #1 %
294 \{\the\xwebModule.\}-%}
295 \let\protect\@unexpandable@protect
296 \edef\next\write\xweb\_cont{%
297 \ZZ{#3}{\#1}{\xweb\_secno}{\thepage}}%
298 \next
299 \message \ZZ{title}{depth}{sec}{page} to .con file
300 \fi
301 \ifnumc@xwebOutlineDepth > #1%
302 \xwebPDFOutline{\xweb\_secno}{#3}% args: nr, title
303 \fi
304 }
305 \end{verbatim}

\section{13.5 Normal sections are started with \texttt{\M{}}.}

\begin{verbatim}
306 \{main\}
307 \def\M{1,}%
308 \ifon\end{xwebModule}\fi
309 \xweb\_PrepareSection{#1}%
310 \ifon\begin{xwebModule}%
311 \ifnumc@xwebDepth > 9%
312 \addcontentsline{lop}{xwebsection}{%
\protect\global\xwebGroupLevel 4 \the\webModule.\% 
\let\protect@unexpandable@protect
\edef\next\write\xweb_content{%
ZZ{4}{#1}{\thepage}%%
\next
}% write "ZZ{title}{depth}{sec}{page}" to .con file
\fi
}
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.

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

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:

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

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.

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

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

\begin{verbatim}
355 (\*main
356 \newdimen\xwebLopIndent
357 \xwebLopIndent=2em
358 \newcount\xwebLopIndentMaxLevel
359 \xwebLopIndentMaxLevel=4 % group level <= max level
360 (/\*main

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

\begin{verbatim}
361 (\*main
362 \def\@starred#1{% % page will be processed later
363 \xwebGroupLevel\z0 % default value of group level
364 \begingroup
365 \let\numberline\@gobble % width not known yet
366 \setbox\z0\hbox{#1}\%
367 \endgroup
368 \ifnum \xwebGroupLevel=\z0
369 \let\next\xweb-BoldTocLine
370 \else
371 \let\next\xweb-NormalTocLine
372 \fi
373 \next{#1}
374 (/\*main

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

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

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

\begin{verbatim}
375 (\*main
376 \def\xweb-NormalTocLine#1{% % page will be processed later
377 \ifnum \xwebGroupLevel>\xwebLopIndentMaxLevel
378 \xwebGroupLevel\xwebLopIndentMaxLevel
379 \fi
380 \advance\xwebGroupLevel\mone % group level -= 1
381 \@dottedtocline
382 \% \xwebGroupLevel % level (old)
383 1 % level
384 {\xwebGroupLevel\xwebLopIndent}% % basic indent
385 \xwebLopIndent % numwidth
386 {#1}% % entry
387 }
388 (/\*main

14 THE LIST OF PROGRAMS.
14.6 A bold line is typeset like a normal line, only the entry is typeset in bold.

```latex
\def\xweb/BoldTocLine/#1/#2/{% 
  \dottedtocline{1}{1cm}{\bf \hfill \hbox to .5em {/#1}}
}
```

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

```latex
\def\xweb/TTLine/#1/#2/{% 
  \addpenalty\secpenalty 
  \addvspace{2em plus.5em}\parfillskip 0pt plus .5em
  \begingroup 
  \noindent
  \hangindent\xweb/TTLine
  \hfill \hbox to .5em {/#1}\nobreak
  \hfill \hbox to .5em {/#2}
  \par
  \endgroup
}
```
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.

\begin{verbatim}
414 \{main
415 \def\PB#1{%
416 \begingroup
417 \xweb_Rprogram
418 \leavevmode
419 #1
420 \endgroup}
421 \}/main
\end{verbatim}

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.

\LaTeX{}ical note: The flag @noskipsec may be used to test if we’re immediately after a run-in section heading.

\begin{verbatim}
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
\end{verbatim}

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.

\begin{verbatim}
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_0ptBreak \5%
444 \xweb_break \6%
445 \xweb_BigBreak \7%
\end{verbatim}
\web_noindent \s%
\% C/C++ tokens
\webRel \?%
\webAddress \AND
\webComplement \CM
\webScope \DC
\webEquiv \E
\webGe \G
\webRightShift \GG
\webNe \I
\webAssign \K
\webLeftShift \LL
\webMod \MOD
\webNull \NULL
\webNot \R
\webBinOr \OR
\webThis \this
\webOr \V
\webAnd \W
\webXor \XOR
\webLE \Z
\webPointer \MG
\webPointerMemberRef \MGA
\webDecr \PP
\% more tokens
\webId \%
\webIdLetter \%
\webRes \&%
\webString \%
\webStringBreak \%
\webNumber \T
\webCombinedOp \MRL
\% goes to TeX state
\webComment \%
\webCxxComment \%
\webRefName \X
\% CWEB tokens
\webMacrosHere \ATH
\webDefine \D
\webFormat \F
\webIdCat \J
\webVerbString \vb
\% cross reference tags
\webChangeFlag \*%
\webCRAiso \A
\webCRAiso \As
\webCRCite \Q
\webCRCite \QS
\webCRUse \U
\webCRsUse \Us
\webCRCite \ET
\webCRsCite \ETS
\% finish the list
\stop\stop
\def\web_UserBindings{%
\web_RestoreBindings
\let\web_UserBindings\relax
}%
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 \web_SaveParskip. A few other layout parameters from the document layout must be saved as well.

508 (/main)

\newskip\web_SaveParskip
\newskip\web_SaveRightskip
\newcount\web_SaveSemSFCode
\newcount\web_SavePretolerance
\newcount\web_SaveHyphenpenalty
\newcount\web_SaveExhyphenpenalty
\newskip\xweb_SaveParskip
\newskip\xweb_SaveRightskip
\newcount\xweb_SaveSemSFCode
\newcount\xweb_SavePretolerance
\newcount\xweb_SaveHyphenpenalty
\newcount\xweb_SaveExhyphenpenalty
\def\xweb_SaveDocLayout {%
\web_SaveParskip\parskip
\web_SaveRightskip\rightskip
\web_SaveSemSFCode\sfcode\string";;" % changed by \web_ProgramLayout
\web_SavePretolerance\pretolerance
\web_SaveHyphenpenalty\hyphenpenalty
\web_SaveExhyphenpenalty\exhyphenpenalty
}
\def\xweb_RestoreDocLayout {%
\parskip\xweb_SaveParskip
\rightskip\xweb_SaveRightskip
\sfcode\string";;"\xweb_SaveSemSFCode
\pretolerance\xweb_SavePretolerance
\hyphenpenalty\xweb_SaveHyphenpenalty
\exhyphenpenalty\xweb_SaveExhyphenpenalty
}
\def\xweb_ProgramLayout {%
\ifx \web_DocLayout\relax
\xweb_DocLayoutInit
\xweb_SaveDocLayout
\def\xweb_DocLayout {%
\let\xweb_DocLayout\relax
}% set new values
\parskip\z@skip
\rightskip\z@ plus 100\p@ minus 10\p@
\sfcode\string":=3000 % same stretch factor as period.
\pretolerance\@M
\hyphenpenalty 9999 % strings can be broken this way
\exhyphenpenalty\@M
\fi
}
(/main)

15.4 When \web_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 \web_DocLayoutInit.

If the program layout is already active, we must not switch it on another time. This can be tested by the current binding of \web_DocLayout.
15.5 The unit of the basic indentation is stored in \texttt{\webIndentUnit}. Continuation lines are indented two units further, i.e., the initial hanging indentation is 3 units. The current hanging indentation is kept in \texttt{\web_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 \texttt{\webIndentUnit}, it’s done in the program state initialization.

\begin{verbatim}
553 \{main\}
554 \newdimen\webIndentUnit \webIndentUnit=1em
555 \newdimen\web_indent
556 \newbox\web_bak \% backspace one unit
557 \newbox\web_bakk \% backspace two units
558 \def\web_LayoutInit{%
559 \global\setbox\web_bak \hbox to \textwidth{-1}\webIndentUnit{%}
560 \global\setbox\web_bakk \hbox to \textwidth{-2}\webIndentUnit{%
561 \global\let\web_LayoutInit\relax
562 }
563 \}

564 \{main\}
565 \def\B{%
566 \if@noskipsec \hspace 1em \fi
567 \web_program
568 \web_indent 3\webIndentUnit \hangindent\web_indent
569 \ifvmode
570 \if@noskipsec
571 \indent \% add an empty documentation part
572 \else
573 \noindent \kern\webIndentUnit
574 \fi
575 \fi
576 }
577 \}

578 \{main\}
579 \def\web_break{% \% forced break, between statements
580 \ifmmode\else
581 \endgraf \% in \LaTeX{} it isn’t sure what \texttt{\par} is really...
582 \noindent
583 \hangindent\web_indent \kern\web_indent
584 \copy\web_bak \% go back to basic indentation
585 \ignorespaces
586 \fi
587 }

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 \texttt{\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. \texttt{\B} can test for this case: \texttt{@noskipsec} is still true then.

\begin{verbatim}
554 \{main\}
555 \newdimen\webIndentUnit \webIndentUnit=1em
556 \newdimen\web_indent
557 \newbox\web_bak \% backspace one unit
558 \newbox\web_bakk \% backspace two units
559 \def\web_LayoutInit{%
560 \global\setbox\web_bak \hbox to \textwidth{-1}\webIndentUnit{%}
561 \global\setbox\web_bakk \hbox to \textwidth{-2}\webIndentUnit{%
562 \global\let\web_LayoutInit\relax
563 }

564 \{main\}
565 \def\B{%
566 \if@noskipsec \hspace 1em \fi
567 \web_program
568 \web_indent 3\webIndentUnit \hangindent\web_indent
569 \ifvmode
570 \if@noskipsec
571 \indent \% add an empty documentation part
572 \else
573 \noindent \kern\webIndentUnit
574 \fi
575 \fi
576 }
577 \}

578 \{main\}
579 \def\web_break{% \% forced break, between statements
580 \ifmmode\else
581 \endgraf \% in \LaTeX{} it isn’t sure what \texttt{\par} is really...
582 \noindent
583 \hangindent\web_indent \kern\web_indent
584 \copy\web_bak \% go back to basic indentation
585 \ignorespaces
586 \fi
587 }

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

An optimal 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.

\begin{verbatim}
578 \{main\}
579 \def\web_break{% \% forced break, between statements
580 \ifmmode\else
581 \endgraf \% in \LaTeX{} it isn’t sure what \texttt{\par} is really...
582 \noindent
583 \hangindent\web_indent \kern\web_indent
584 \copy\web_bak \% go back to basic indentation
585 \ignorespaces
586 \fi
587 }

553 \{main\}
554 \newdimen\webIndentUnit \webIndentUnit=1em
555 \newdimen\web_indent
556 \newbox\web_bak \% backspace one unit
557 \newbox\web_bakk \% backspace two units
558 \def\web_LayoutInit{%
559 \global\setbox\web_bak \hbox to \textwidth{-1}\webIndentUnit{%}
560 \global\setbox\web_bakk \hbox to \textwidth{-2}\webIndentUnit{%
561 \global\let\web_LayoutInit\relax
562 }
563 \}

564 \{main\}
565 \def\B{%
566 \if@noskipsec \hspace 1em \fi
567 \web_program
568 \web_indent 3\webIndentUnit \hangindent\web_indent
569 \ifvmode
570 \if@noskipsec
571 \indent \% add an empty documentation part
572 \else
573 \noindent \kern\webIndentUnit
574 \fi
575 \fi
576 }
577 \}

578 \{main\}
579 \def\web_break{% \% forced break, between statements
580 \ifmmode\else
581 \endgraf \% in \LaTeX{} it isn’t sure what \texttt{\par} is really...
582 \noindent
583 \hangindent\web_indent \kern\web_indent
584 \copy\web_bak \% go back to basic indentation
585 \ignorespaces
586 \fi
587 }

\end{verbatim}
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.

\[h = \text{main i}\]

\[h = \text{main i}\]
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.

\begin{verbatim}
(*main*)
\let\xwebAnd=\land % logical and, &
\let\xwebEquiv=\equiv % equiv sign, for ==
\let\xwebGe=\ge % greater or equal
\let\xwebLe=\le % less or equal
\let\xwebNe=\ne % unequal, !=
\let\xwebNull=\lambda % NULL pointer
\let\xwebNot=\lnot % logical not, !
\let\xwebOr=\lor % logical or, |
\let\xwebXor=\oplus % bitwise exclusive or, ^
\end{verbatim}

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 10pt fonts. We use the plain \TeX{} names for these fonts since we need them especially, it’s not possible to substitute them without changing these macros. If the NFSS is used with \TeX{} 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.

\begin{verbatim}
(*main*)
\font\teni=cmr10
\font\sevensy=cmss7
\newbox\xweb_pointer % pointer to struct component, ->
\setbox\xweb_pointer\hbox{\kern-2pt\lower3pt\hbox{\teni\char'176}}\kern1pt
\newbox\xweb_decr % decrement, --
\setbox\xweb_decr\hbox{%
  \kern .5pt
  \raise 1pt \hbox{\sevensy\char0 \kern-.5pt\char0}%
  \kern .5pt
}
\newbox\xweb_incr % increment, ++
\setbox\xweb_incr\hbox{%
  \kern .05em
  \raise .1em \hbox{\scriptstyle {+}}\kern -.05em{+}\
  \kern .05em
}
\end{verbatim}

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 . . .).

\begin{verbatim}
(*main*)
\let\xwebMod=%
\end{verbatim}

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 . . .
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 `\webRes` 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?

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

Within strings certain cseqs have a special meaning; this is introduced by `\web_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 `\webStringBreak`. This break is shown with the C convention of escaped newlines.
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.

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

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 2quad in front if the line is not broken and 1.5 quad if the line is broken.
717 \%$
718 $\backslash$,\texttt{ast/$\%$
719 \}$
720 \texttt{\let\xwebCommentHook\relax}
721 \texttt{\let\xwebCxxComment\xwebComment}
722 $\langle$/main$\rangle$
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').

\begin{verbatim}
(*main)
def\xwebIdCat{\xwebString{@&}}
def\xwebMacrosHere{
 \begingroup
 \def\xwebRefNumber/#/#/1/{/}/
 \xwebRefName :\xwebRefMacrosHere
 \endgroup
}
def\xwebRefMacrosHere{Preprocessor Definitions}

(*main)
\end{verbatim}

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

\begin{verbatim}
(*main)
def\xwebVerbString/#/1/{/{/\fboxsep/\tw/@/\p/@ /\fbox/\xwebString/#/1/}/}/

(*main)
\end{verbatim}

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.

\begin{itemize}
\item One could think about handling special values of the section numbers differently. E.g., an empty argument is not typeset at all, a 0 triggers a marginal note about a missing definition, etc.
\end{itemize}

17.4 A refinement name may be typeset in both 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 \texttt{\xweb\_ToggleText}.

\TeX{}ical note: If we're in math mode \texttt{\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 \texttt{\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 \texttt{	extbackslash}, 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 \texttt{	extbackslash} with \texttt{\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)

\begin{itemize}
\item 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?
\end{itemize}
17.5 The \webRefNumber macro is used in the module names in program text; It switches to footnotesize and calls \webSetModuleNumber to create the reference text.

17.6 There are two macros to typeset module numbers: \webSetModuleNumber takes one number, and \webSetModNrList takes a comma-separated list that is terminated by a period. These macros must employ \the\webModule to get consistent numbers with nonstandard definitions of \the\webModule. The value of the \webModule 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.

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 cw eb macros modified by Andreas Scherer), insert \web_TagSec before each number, filter things like \ET, and then expand the list.
17.8 By defining \webTagSec to \webSetModuleNumber we can now employ \maketoks in \webSetModNrList to typeset a list of references.

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

\begin{verbatim}
806     \else
807     \global\let\next\relax
808     \fi
809     \endgroup
810     \next \#1
811 }
812 ///main
\end{verbatim}
18 Section cross references and changes.

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

Within a list of \( n \) section numbers the first \( n - 1 \) numbers are separated by commas with one following blank. The last two numbers are separated by \( \&T \) if \( n = 2 \), and by \( \&T s \) 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 \texttt{\webNumberListHangindent}. But beware: This isn’t a dimension register, it’s a macro. This way one can use ems as dimensions.

We must assert that we’re in CR state while we set a cross reference. The first parameter of a cross reference information unit is the introductionary text, the second is the number list. The parameters must be evaluated in a group—local parameter changes therein must not influence the environment. The list must be passed to \texttt{\webSetModNrList} to typeset (and hyperreference) them correctly.

18.2 Well, let’s define all introducing tags, which in fact start the cross reference information unit. The number list is gathered by \texttt{\webCrossRef}.

18.3 The changeflag is set as a star.
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 \webJobname.idx, the latter in the file \webJobname.scn. For each of these files exists a flag named xweb_ezEntries (where ext ∈ \{idx, scn\} is the extension of the appropriate file) which tells if there are any entries of the respective category.

\begin{verbatim}
\newif\ifxweb_idxEntries
\newif\ifxweb_scnEntries
\ifx xweb_idxEntries true\ifx xweb_scnEntries true
\endif\fi
\if\ifeof\xweb_TmpFile
\let\next/\defpar
\message{No #1 file present for \webJobname !}
\else
\read\xweb_TmpFile to \next
\fi
\csname xweb_#1Entries\fi
\endcsname
\closein\xweb_TmpFile
\end{verbatim}

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.

\begin{verbatim}
\xweb_TmpFile
\def\xweb_HasEntries#1{%
\openin\xweb_TmpFile \webJobname.#1\relax
\ifeof \xweb_TmpFile
\let\next/\defpar
\message{No #1 file present for \webJobname !}
\else
\read\xweb_TmpFile to \next
\fi
\csname xweb_#1Entries\fi
\endcsname
\closein\xweb_TmpFile
\}
\end{verbatim}

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.

\begin{verbatim}
\xweb_cont
\let\ZZ=\let % now you can \write the control sequence \ZZ
\def\xweb_contentsline#1#2#3#4{%
\ifnum#2=0 \smallbreak\fi
\hbox to \textwidth{\xweb_consetup(#2)#1}
\end{verbatim}
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 `\jobname`s: 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).
including the web file.

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 reflist outside the context of a web document.

(main)
\def\xwebInputIndex#1{\
  \def\setname##1##2\%/\
  \edef\xwebJobname{##1}\
  \ifx##2\%/\
  \edef\ext{idx}\
  \else\
  \edef\ext{\chop##2\%/}\
  \fi\
  \fi\
  \def\chop##1\%/\
  \expandafter\setname##1\%/\xwebIdIndex{\xwebJobname.##1}\
  \else\
  \edef\ext{\chop##2\%/}\
  \fi\
  \fi\
  \xwebIdIndex{\xwebJobname.\ext}\
}\

(main)
\def\xwebInputRefList#1{\
  \def\setname##1##2\%/\
  \edef\xwebJobname{##1}\
  \ifx##2\%/\
  \edef\ext{scn}\
  \else\
  \edef\ext{\chop##2\%/}\
  \fi\
  \fi\
  \def\chop##1\%/\
  \expandafter\setname##1\%/\xwebRefList{\xwebJobname.##1}\
  \else\
  \edef\ext{\chop##2\%/}\
  \fi\
  \fi\
  \xwebRefList{\xwebJobname.\ext}\
}\

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

\begin{verbatim}
967 {main}
968 \def:\ifstr#1#2#3{%
969 \edef:\web\tempa(#1)%
970 \edef:\web\tempb(#2)%
971 \if\web\tempa\web\tempb #3\fi
972 }
973 {/main}
\end{verbatim}

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

\begin{verbatim}
974 {main}
975 \def:\webfile{%\@ifnextchar%{
976 \{\xweb\_file\}{\xweb\_file[]}}
977 {/main}
\end{verbatim}

19.9 The square-braced argument is mandatory for \web_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, \web\_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).

\begin{verbatim}
978 {main}
979 \newif:\if\opt
980 \def:\web\off{%\let:\maybe=:\iffalse}
981 \def:\web\on{%\let:\maybe=:\iftrue}
982 \def:\web\_file[#1][#2]{
983 \ifx:\web\GlobalIndex\web\Indextrue\else\web\Indexfalse\fi
984 \ifx:\web\GlobalRef\web\Refture\else\web\Reffalse\fi
985 \ifx:\web\GlobalRagged\web\Raggedtrue\else\web\Raggedfalse\fi
986 \ifx:\web\GlobalSec\web\off\else\web\on\fi
987 \ifx:\web\GlobalCon\web\Condtrue\else\web\Confalsete\fi
988 \ifx:\web\GlobalHideFormats\web\HideFormatstrue\else
989 \web\HideFormatsfalse\fi
990 \xweb\_hypertype=\xweb\_GlobalHypertype
991 \@for:\web\opt:=#1\do{
992 \@optfalse
993 \if\opt\false
994 \ifstr:\web\opt{index}{\web\Indextrue:\@opttrue}
995 \ifstr:\web\opt{noindex}{\web\Indexfalse:\@opttrue}
996 \ifstr:\web\opt{reflist}{\web\Refture:\@opttrue}
997 \ifstr:\web\opt{noreflist}{\web\Reffalse:\@opttrue}
998 \ifstr:\web\opt{raggedbottom}{\web\Raggedtrue:\@opttrue}
999 \ifstr:\web\opt{flushbottom}{\web\Raggedfalse:\@opttrue}
1000 \ifstr:\web\opt{onlychanges}{\web\off:\@opttrue}
1001 \ifstr:\web\opt{allsections}{\web\on:\@opttrue}
1002 \ifstr:\web\opt{nocon}{\web\Confalse:\@opttrue}
1003 \ifstr:\web\opt{contents}{\web\Condtrue:\@opttrue}
1004 \ifstr:\web\opt{hideformats}{\web\HideFormatsfalse:\@opttrue}
1005 \ifstr:\web\opt{showformats}{\web\HideFormatsfalse:\@opttrue}
1006 \ifstr:\web\opt{hyperref}{\xweb\_hypertype=1:\@opttrue}
1007 \ifstr:\web\opt{pdftex}{\xweb\_hypertype=2:\@opttrue}
1008 \ifstr:\web\opt{nohype}{\xweb\_hypertype=0:\@opttrue}
1009 \if\opt\false\else\PackageError{webfiles}{%
1010 Unknown option for the \protect\webfile\space command
\end{verbatim}
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.

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

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 boxes 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.

The following text is set as the header of the index.
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 (``) would be set in one column. It isn't necessary too, we are in two columns already.

\begin{verbatim}
\Text{main}
\newif{\ifxweb_multicol} \xweb_multicolfalse
\IfFileExists{multicol.sty}{
  \if@twocolumn
  \xweb_multicoltrue
  \else
  \fi
}{
}{
\newenvironment{xweb_twocolumn}{
\begin{multicols}{2}\}
}{
\end{multicols}\}
\else
\newenvironment{xweb_twocolumn}{
\if@twocolumn
\columnsep 3.5\p@ \%
\fi
\begin{onecolumn}\}
\else
\fi
\Text{main}\end{verbatim}

19.14 Now `\xwebIdIndex` itself. PDFTeX will add an outline entry for the index. vfill before `\xweb_IndexTop` is still not strong enough! A pagebreak can occur just after it.

\begin{verbatim}
\Text{main}
\def\xwebJobName{noname}
\def\xweb_SetupIndex{%
\begin{xweb_twocolumn}[\xweb_IndexTop]
\ifnum\xweb_hypertype=2
  \pdfdest name{\xwebJobname:index} \%
  \pdfoutline goto name{\xwebJobname:index}{Index}
\fi
% go to CR mode\begin{verbatim}
\end{verbatim}
\Text{main}\end{verbatim}

\begin{verbatim}
\Text{main}
\end{verbatim}

\begin{verbatim}
\Text{main}
\end{verbatim}
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 \texttt{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 \texttt{\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.)

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

19.17 The list of the refinement names is available in the file \texttt{\xweb\texttt{\jobname}.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 \texttt{\parfillskip} and \texttt{*} since it was changed in the index.
The redefinition of \webCrossRef etc. must be done locally to prevent troubles with trailing web files: Crossref info in these files would be indented!
20 Bells and whistles.

\(\wedge\) Just copied from the plain version. It doesn’t work anyhow—\texttt{\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.

\begin{verbatim}
\catcode`\_/=12 \postATL} % print $1 in limbo
\def\postATL/#1/#2/{
  \bf letter \texttt{\uppercase{\char/#1}}
}\def\noATL/#1/#2/{}
\def\noatl/\let/ATL=/noATL/ % suppress output from \ATL
\ProcessOptions
\ifxweb\_multicol
  \RequirePackage{multicol}
\fi
\catcode`\_/=\xwebCatUsCode
\endinput
\end{verbatim}

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

\begin{verbatim}
\ProcessOptions
\ifxweb\_multicol
  \RequirePackage{multicol}
\fi
\catcode`\_/=\xwebCatUsCode
\endinput
\end{verbatim}
21 Spidery Webs.

The file _webbind.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 "\webbind.sty" or such.

21.1 First, make the underscore usable in private control sequences, as in _webfiles.sty_:

```latex
\catcode`\_=`\webCatLetter
\catcode`@=`\webCatLetter
```

21.2 Some miscellaneous...

```latex
\let\&=\amp \let\$=\SS \let\P=\PP
```

21.3 The main section tag will be a mixture of _webfiles_ and _webkernel.tex_:

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

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.
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
  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

@*/ = part, don’t skip page.
21.6 Surround text in vertical bars: (already tagged with \PB in Spider 3.0)

\def\CD#1\DC{#1}

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

\def\P{\B}
\def\\P{\B}

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

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

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

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

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

21.15 Restore catcodes:

References


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 | 106 | 1244 |
|------------------------------------------------|
| \\& | 126 | 1244 |
| \) | 126 | 1244 |
| \/* | 216 | 126 | 1244 |
| \_ | 126 | 1244 |
| \? | 126 | 1244 |
| @M | 398, 539, 541, 1072 |
| @currentlabel | 251 |
| @defpar | 851, 857 |
| @dottedocline | 373, 383 |
| @flushglue | 1120 |
| @for | 983 |
| @gobble | 331, 344, 357, 793 |
| @ifnextchar | 967 |
| @input | 1084, 1133 |
| @mbfboth | 319, 322 |
| @noskipsecfalse | 255 |
| @oskipsectrue | 258 |
| @optfalse | 985 |
| @opttrue | 986–1000 |
| @pnumwidth | 400 |
| @secpenalty | 392, 416 |
| @starttoc | 320, 323 |
| @topma | 1193, 1198 |
| @toppb | 1193, 1199 |
| @toppco | 1198, 1201 |
| @toppdo | 1198–1200 |
| @toppdim | 399 |
| @toppe | 1193, 1198 |
| @tocnarg | 397 |
| @unexpandableprotect | 287, 306, 1182, 1331 |
| \ | 126 | 1244 |
| \0 | 1244 |
| \1 | 126 | 1244 |
| \2 | 126 | 1244 |
| \3 | 126 | 1244 |
| \4 | 126 | 1244 |
| \5 | 126 | 1244 |
| \6 | 126 | 1244 |
| \7 | 126 | 1244 |
| \8 | 126 | 1244 |
| \9 | 1096 |

| \A | 426 | 1244 |
| \addcontentsline | 284, 304, 908, 1180, 1327 |
| \addpenalty | 392 |
| \addtoks | 756 |
| \addvspace | 393, 403 |
| \aftergroup | 687, 688, 695 |
| \amp | 1156 |
| \anchor | 193, 194 |
| \AND | 126 | 1244 |
| \arabic | 171, 237 |
| \As | 426 | 1244 |
| \ast | 705, 710 |
| \ATH | 426 | 1244 |
| \ATL | 1137 |
| \B | 556, 1241, 1242 |
| \bgroup | 250, 694, 1138 |
| \Black | 202 |
| \C | 426 | 1244 |
| \@xwebDepth | 90, 91, 283, 303, 907, 1179, 1326 |
| \@xwebModule | 191, 745 |
| \\c@xwebOutlineDepth | 293, 1189 |
| \@xwebSecNoEject | 274 |
| \CD | 1237 |
| \chapterline | 910 |
| \ch | 917 |
| \con | 916, 1339 |
| \cwebfile | 970 |
| \D | 426 | 1244 |
| \Declarations | 426 | 1238, 1244 |
| \DeclareOption | 69–80, 84–86, 90, 91 |
| \discretionary | 684 |
| \do | 417 |
| \do | 983 |
| \DoInput | 59 |
| \documentclass | 2, 59 |
| \E | 426 |
| \egroup | 262, 694, 1140 |
| \EndCrossrefs | 11 |
| \end | 32, 268, 300 |
| \endgraf | 573 |
| \endinput | 1150 |
| \environmenst | 24 |
| \equiv | 608 |
| \ET | 426 | 1244 |
| \F | 426 | 1244 |
| \fbox | 726 |
| \fboxsep | 726 |
| \filbreak | 248 |
| \fin | 915, 1339 |
| \first | 700, 703–706, 771, 774 |
| \font | 618, 619 |
| \footnotesize | 645, 740, 821, 1123 |
| \G | 426 | 1244 |
| \ge | 609 |
| \GG | 426 | 1244 |
| \GG | 650 |
| \H | 396, 560, 575, 598, 822, 1091, 112 |
| \hfil | 581, 591, 870, 891 |
| \hfill | 400 |
| \hfilneg | 352, 592 |
| \hrule | 661 |
| \hspace | 1071 |
| \hskip | 261, 282, 558, 603, 704, 875–880, 1173, 1354 |
| \hspace | 259 |
| \hss | 400, 872, 892 |
| \hyperlink | 177 |
| \hypertarget | 178 |
| \hyperpenalty | 514, 522, 540, 1072 |
| \I | 1074, 1122, 1244, 1342, 1347 |
| \if@noskipsec | 254, 419, 558, 562 |
| \if@opt | 971, 1001 |
| \ifcase | 873 |
| \iffalse | 220, 972 |
| \ifFileExists | 884, 1042 |
| \ifmode | 572, 730 |
| \ifnum | 173, 176, 192, 273 |
| \iffont | 274, 283, 293, 303, 360, 369, 770, 868, 887, 907, 1059, 1113, 1179, 1189, 1346 |
| \iffont | 216, 268, 280, 300, 302, 919, 1163, 1171, 1323, 1325 |
| \ifstr | 959, 986–1000 |
| \ifttrue | 220, 973 |
| \ifXweb | 561 |
| \ifXweb | 1193 |
| \ifXwebFirstModule | 236 |