DocBy.TE\hbox{X} – Making a Documentation Of Sources By TE\hbox{X}

Petr Olšák
www.olsak.net/docbytex.html

Table of Contents

1 Preface ................................................................. 3

2 For Users ............................................................. 4
  2.1 File Types .......................................................... 4
  2.2 An Example of the Module Documentation .................... 4
  2.3 What Version of TE\hbox{X} for DocBy.TE\hbox{X}? ............... 6
  2.4 Searching Words by EncTE\hbox{X} ............................... 6
  2.5 The Index, Table of Contents, Footnotes and Bookmarks Generation .... 7
  2.6 Source Code Inserting .......................................... 7
  2.7 References to Line Numbers .................................... 9
  2.8 Verbatim Environment by \begtt/\endtt and by Quotes .......... 9
  2.9 The Declaration of the Documented Word ...................... 9
  2.10 Namespaces ..................................................... 11
  2.11 The Application Level of the Documentation ................ 11
  2.12 Title, Parts, Sections, Subsections .......................... 12
  2.13 Hyperlinks, References ........................................ 12
  2.14 Pictures Inserting .............................................. 13
  2.15 Items .......................................................... 13
  2.16 \begtt and \ins ................................................ 13

3 For Advanced Users .................................................. 13
  3.1 Internal Names .................................................. 14
  3.2 Hooks .......................................................... 14
  3.3 The Commands \module and \ins ............................... 15
  3.4 The Comments Turned to Green Color ........................ 15

{version May 2014}
Table of Contents

4 For Designers

4.1 Parameters and Auxiliary Macros .................................................. 16
\size ... 16, \vsize ... 16, \width ... 16, \bbf ... 16, \btt ... 16, \tsize ... 16, \rsize ... 16, \pfont ... 16, \sml ... 16, \tsml ... 16, \ptm ... 16, \snormal ... 16, \bblue ... 16, \Red ... 17, \Brown ... 17, \Green ... 17, \Yellow ... 17, \Black ... 17, \setmycolor ... 17, \osmBlack ... 17, \rectangle ... 17, \docbytex ... 17

4.2 Sections and Subsections ............................................................... 17
\printsec ... 17, \printsecbelow ... 17, \printsubsec ... 18, \printsubsecbelow ... 18, \printpart ... 18, \printpartbelow ... 18, \emptynumber ... 18

4.3 The Title, The Author ....................................................................... 18
\title ... 18, \titltitle ... 18, \projectversion ... 19, \author ... 19

4.4 Headers and Footers .......................................................................... 19
\footline ... 19, \headline ... 19, \normalhead ... 19, \noheadline ... 19, \headtitle ... 19, \headlinebox ... 19

4.5 Printing of the Hyperlink Destinations and Footnote References ........... 20
\printdg ... 20, \printtitleside ... 20, \printfootnote ... 20

4.6 The Index and Table of Contents Item ............................................... 20
\ptocline ... 20, \ptocsubline ... 20, \myeditfill ... 21, \ptocentry ... 21,
\myidots ... 21, \printindexentry ... 21, \separating ... 21

4.7 The Source Code Listing .................................................................. 21
\printiabove ... 21, \printiline ... 21, \printibelow ... 21, \specrule ... 22,
\isnameprinted ... 22

4.8 The \begtt ... \endtt Printing ......................................................... 22
\printvabove ... 22, \printvline ... 22, \printvbelow ... 22

4.9 Pictures ............................................................................................ 22
\figwidth ... 22, \figfig ... 22, \figdir ... 23

4.10 Items ............................................................................................... 23
\begitems ... 23, \enditems ... 23, \itemno ... 23, \dbitem ... 23, \item ... 23

5 For \TeX Wizards

5.1 Auxiliary Macros .............................................................................. 23
\dbtversion ... 24, \lctesttable ... 24, \wordbuffer ... 24, \nactive ... 24,
\emptyv ... 24, \words ... 25, \onlyactive ... 25, \vword ... 25

5.2 Initialization ....................................................................................... 24
\dbtversion ... 24, \lctesttable ... 24, \wordbuffer ... 24, \nactive ... 24,
\emptyv ... 24, \words ... 25, \onlyactive ... 25, \vword ... 25

5.3 The \firstline, \-line, \-label Macros ................................................... 25
\lineno ... 25, \-lineno ... 25, \firstline ... 25, \-fcontinue ... 25, \-fskipping ... 25,
\tskippingfalse ... 25, \tskippingtrue ... 25, \first ... 25, \inputfilename ... 25,
\next ... 26, \-osword ... 26, \readparamwhy ... 26, \-startline ... 26, \stopline ... 26,
\scanparam ... 26, \scanparam ... 26, \scanparam ... 26, \scanparam ... 26,
\insinternal ... 26, \-testline ... 27, \-continue ... 27, \returns ... 27,
\readneiline ... 27, \printline ... 28, \-lastline ... 28, \-label ... 28,
\label ... 28, \-label ... 28, \-testlabel ... 28

5.4 Commands \begtt, \endtt .............................................................. 28
\begtt ... 28, \-startverb ... 28, \runttt ... 28, \-endtt ... 28,
\scannexttoken ... 28

5.5 The Namespaces ............................................................................... 29
\namespacemacro ... 29, \namespace ... 29, \locword ... 29, \endnamespace ... 29,
\rewritten ... 30, \word ... 30, \genlongword ... 30, \refts ... 30, \refts ... 30,
\curs ... 30

5.6 The \dg Command and Friends ....................................................... 30
1 Preface

DocBy.TEX gives you a possibility to creating a documentation of source codes by T\TeX. The source codes can be \textit{i} C language or whatever other computer language.

On the contrast of Knuth’s “literal programming” this tool does not use any preprocessors for doing filters of information for human and for computer which is stored in single source file. I suppose that programmers prefer to write and tune the program in computer language before they start to write the documentation. It would be fine to write the documentation after that and without modifying of the source code of the working program. Modern systems gives possibility to open more windows with more than one text editors: you can see the source code in one editor and write the documentation of the source code of the working program. Modern systems gives possibility to open more windows with more than one text editors: you can see the source code in one editor and write the documentation of the source code of the working program. Modern systems gives possibility to open more windows with more than one text editors: you can see the source code in one editor and write the documentation of the source code of the working program.

The first part of this document (2) describes the DocBy.TEX at user level. The next part documents the implicit macros implemented in DocBy.TEX, which are supposed that experienced user will want to change them in order to realize special wishes. The next section 4 includes the documentation of design-like macros. User can change them to create a better look of his/her document. The last section 5 describes all macros of DocBy.TEX at implementation level in detail.

This document is created by DocBy.TEX itself, it means that it can serve as an example of DocBy.TEX usage.
2 For Users

2.1 File Types

The DocBy.Tex is proposed as a tool for making documentation of C language. That is a reason why the next example is a documentation of the hypothetical program written in this language. If you need to document another computer language, you can change some macros (see the section 3).

We suppose that the source code is separated into “modules”. Each module is intended to one special problem which is solved by programmer. Each module has its own name (foo for example) and it is written in files foo.h and foo.c. These files are compiled into foo.o. All modules are linked at the end of compilation into the executable program.

If we want to document these source files, we create new file with .d extension for each module, for example foo.d. The documentation of the module will be written in that file. Next we create the main file (for example program.tex) where all *.d files are included by the command \module. You can use commands \title (name of the program), \author (name of the author) and (for example) \dotoc for making of table of contents, \doindex for generating of the index. Of course, you can write first or general notes to the program in the main file too. The contents of the file program.tex can be:

\input docby.tex
\title The Program lup -- Documentation of The Source Codes
\author Progr and Ammer
\dotoc % the table of contents will be here
\sec The structure of the source files

The source files are in the three modules.
The auxiliary functions are defined in "base.c" and "base.h" files.
The window management are solved in "win.c" and "win.h" files.
The file "main.c" includes the function "main".
\module base
\module win
\module main
\doindex % the index will be created here
\bye

We decided to sort the documentation from “simple” functions to the more complicated problems. Somebody can prefer another way from main function first and the auxiliary functions at the end. He/she can write:

\module main
\module win
\module base
\doindex
\bye

Both ways are possible because the documentation is hyperlinked automatically. When the reader see the usage of some function, he/she can simply go to the definition of this function simply by one click. The reverse hyperlinks are included too.

2.2 An Example of the Module Documentation

Let we document the module foo in the file foo.d. This file is included by module foo command.

We can document any part of source foo.c by words and combine this by a listing of parts of source foo.c or foo.h by command \ins c⟨keyword⟩ or \ins h⟨keyword⟩. The part of the source code is declared usually by // c⟨keyword⟩ line. The example follows.

Suppose that the following text is written in the file foo.d
The struct \texttt{mpair} is used as a return value of "\texttt{my\_special\_function}". There are two "\texttt{float}" values.

\begin{verbatim}
\struct\ mypair

The \texttt{mpair struct} \texttt{my\_special\_function()} has one parameter "\texttt{p}" and returns double and triple of this parameter in "\texttt{mpair struct}".

\begin{verbatim}
\struct\ mypair

The file \texttt{foo.c} has to include the comments //: \texttt{mpair} and //: \texttt{my\_special\_function}. These comments delimit the part of source code to be listed in the documentation:

\begin{verbatim}
#include <stdio.h>

//: \texttt{mpair}
struct mypair {
  float x, y;
};

//: \texttt{my\_special\_function}
struct my_special_function (float p)
{
  struct mypair return_pair;
  return_pair.x = 2*p; // double of p
  return_pair.y = 3*p; // triple of p
  return return_pair;
}
\end{verbatim}

The result looks like that:

The struct \texttt{mpair} is used as a return value of \texttt{my\_special\_function}. There are two \texttt{float} values.

\begin{verbatim}
foo.c
5: struct mypair {
6:   float x, y;
7:    }

The \texttt{my\_special\_function} has one parameter \texttt{p} and returns double and triple of this parameter in \texttt{mpair} struct.

\begin{verbatim}
foo.c
11: struct my_special_function (float p)
12: {
13:   struct mypair return_pair;
14:   return_pair.x = 2*p; // double of p
15:   return_pair.y = 3*p; // triple of p
16:   return return_pair;
17: }
\end{verbatim}

The first listed part of source code is started by //: \texttt{mpair} and ended by first occurrence of the //:. The second listed part is started by //: \texttt{my\_special\_function} and ended at the end of file. These delimiters (and the neighbouring empty lines) are not printed.

The order of the listed parts are independent of the order in source file. We can first comment my special function and include its source code. Afterward we can explain the structure mypair and show the source code of this structure.

Notice that the numbers of lines are inserted exactly by the lines in source code. It means that the missing line \texttt{#include <stdio.h>} has number one and first printed line has the number five.

The //:(\texttt{keyword}) delimiter and the closing delimiter //: can be at arbitrary place of the line, no essential at begin of line. The lines with the delimiters are not printed.

\texttt{struct mypair: 5} \quad \texttt{struct mypair my\_special\_function(): 5}
Notice the command \dg in source of the documentation. The documented word (separated by space) follows immediately. The optional parameter in brackets is interpreted as “type” of the documented word. The documented word is printed in red color on the rectangle and all occurrences of that word in the documentation is printed in blue color and treated as hyperlink to the place where is the word documented (red color). The occurrence of that word have to be written between the quotes "..." or it is placed in the inserted source code. You need not do any marks in source code in order to highlight the usage of the documented word. This is done automatically.

If the documented word has the brackets ( at the end, then it is the function. These brackets are not printed in the current place, but they are printed in the footnotes and in the index.

The quotes "..." are delimiters of “parts of listings inside paragraph”. This text is printed by typewriter font and the occurrences of documented words are hyperlinked here. All characters have printed here without re-interpretation, it means this environment behaves like “verbatim”.

The footnote includes a list of all documented words on the current page. Each word is followed by list of pages here. These pages points to all pages here the documented word occurs.

All documented words are automatically inserted to the alphabetical index created by \doindex command.

### 2.3 What Version of \TeX for DocBy.\TeX?

In order to activate all features mentioned above we need to use pdf\TeX extended by enc\TeX. The language of automatically generated words (such as Contents, Index) is selected by current value of \language register when \input docby.tex is processed. DocBy.\TeX writes on the terminal the “modes” information:

```
This is DocBy.TeX, version May 2014, modes: enc+PDF+ENG
```

DocBy.\TeX can work in the following modes: enc/NOenc, PDF/DVI, ENG/CS.

The enc mode is activated if the enc\TeX is detected. Otherwise (if enc\TeX is unavailable), DocBy.\TeX prints warning and sets the NOenc mode: the occurrences of documented words are not detected and hyperlinked. The index is much more poor, because the pages with occurrences of the words are missing. Only the places of documentation of the words are referred. It means that the enc\TeX extension is very important for DocBy.\TeX. This extension is usually available in current \TeX distributions and it is activated by pdfcsplain format. So the recommendation is: use pdfcsplain when you are using DocBy.\TeX.

The PDF mode is activated if the pdf\TeX is used. Otherwise DocBy.\TeX switches to the DVI mode and prints the warning message on the terminal. The colors and hyperlinks are not working in DVI mode but the list of pages with all occurrences of documented words is printed in index (if enc\TeX is activated).

If \language=0 or (pdf)csplain isn’t used then language mode is set to ENG (English words will be generated). Else this mode is set to CS (Czech words will be generated). If you are using another language, you need to redefine some macros, see section 3.1.

### 2.4 Searching Words by Enc\TeX

The hyperlinked words are located by enc\TeX by “hungry algorithm”. It means that if there are two documented words abc and abcdde then the text abcd\textcolor{blue}{efg} is divided to the hyperlinked part abcd\textcolor{blue}{e} (the blue color is used) and to the normal part \textcolor{black}{fg} (black color). The hyperlinked part points to the place of the documentation of the word \textcolor{blue}{abcd}e. On the other hand the text \textcolor{black}{abcdx} is divided to hyperlinked part \textcolor{blue}{abc} and this part points to the documentation of the word \textcolor{blue}{abc}.

Enc\TeX is not able to work with regular expositions. It means that there is no simple possibility to search only words bounded by spaces, other white characters or by punctuation. Enc\TeX searches the word as a part of another word. This leads to unexpected situations: the short word is documented but it is a part of longer undocumented words used in source code. For example, you document the structure turn but you don’t need to hyperlink the part of the word return. In such case you can define the return word as a “normal” undocumented word by the command \noactive\{word\} (for example \noactive\{return\}). This command declares the \textcolor{blue}{word} as a searched word (for enc\TeX) but sets it as inactive.

Imagine that you document a word which is used in code in “documented meaning” only if some text precedes this word and/or some text followed the word. If the word is used with another
prefix/postfix then this is undocumented meaning of the word. You can use in such case a declaration `\onlyactive{⟨before⟩}{⟨word⟩}{⟨post⟩}`. If you declare the word by `\dg_{⟨word⟩}` (or by similar manner, see section 2.9), then the word is hyperlinked in source code only if the text ⟨before⟩ precedes and the text ⟨post⟩ follows. The text ⟨before⟩ and/or ⟨post⟩ itself stays inactive. The parameters ⟨before⟩ or ⟨post⟩ can be empty (no both simultaneously) and you can use more `\onlyactive` declarations of single ⟨word⟩.

DocBy.TeX activates the encTeX searching only inside the group "..." or in listings of source codes. It means that `\mubytein=1` (see encTeX documentation) is set only in these situations. We recommend to leave `\mubytein=0` outside these environment. If you set `\mubytein=1` (for example because of UTF-8 encoding) for the whole document then you do it on your own risk. The words inside your comments can be hyperlinked in such case.

2.5 The Index, Table of Contents, Footnotes and Bookmarks Generation

The index and table of contents generation is fully done on macro level of DocBy.TeX. You needn’t use any external program (DocBy.TeX itself does the alphabetical sorting). Just write `\doindex` or `\dotoc` on the desired place in your document. Warning: the table of contents is not correctly generated after first pass of TeX. You have to run TeX twice. The pages may be changed after second pass because of table of contents is inserted. Thus correct output is (may be) guaranteed after third pass of TeX. The words “may be” are written here due to the problem with footnotes mentioned in section 5.7. The footnotes are changed in all three TeX runs and this influences the vertical typesetting retrospectively. This is a reason why DocBy.TeX performs the check of consistency of references generated by current and previous TeX pass. This check is done during the `\bye` macro is processing. Thus, it is usable to write `\bye` command instead `\end` primitive command at the end of the document. If the `\bye` macro is used then you can see the message “OK, all references are consistent” on the terminal or the warning “page references are inconsistent, run me again”.

You can do test of consistency in more detail by following script:

```bash
#!/bin/bash
cp document.ref document.r0
pdfcsplain document
diff document.r0 document.ref
```

DocBy.TeX tries to fix the footnote processing after second pass in order to document convergence. If you do big changes in the document after that then DocBy.TeX does change the numbers of lines for footnotes and the Overfull/Underfull boxes may occur. We recommend to remove the .ref file and to run three passes of DocBy.TeX again in such case.

DocBy.TeX creates the structured bookmarks in PDF output if `\bookmarks` command is used. The structured bookmarks include names of parts, sections, subsections and documented words. There is no matter where the command `\bookmarks` is written because the information used in bookmarks is read from .ref file. The problem about encoding of texts of bookmarks is discussed in section 3.2.

2.6 Source Code Inserting

Instead of simply command `\ins` you can use two more elaborate commands `\ifirst` and `\inext` in order to insert a part of source code in your documentation.

The `\ifirst{⟨file⟩}{⟨from⟩}{⟨to⟩}{⟨why⟩}` command inserts a part of the file ⟨file⟩ (full file name including extension) from first line with the pattern ⟨from⟩ ending by line with the pattern ⟨to⟩ or (if such line does not exists) to the end of file. If the pattern ⟨from⟩ does not exists then the warning is printed on the terminal.

The parameters of `\ifirst` command are first expanded and used thereafter. The active tie character is expanded to the space.

The parameter ⟨why⟩ specifies if the line with ⟨from⟩ pattern and/or the line with ⟨to⟩ pattern have to be printed or not. This parameter has only two characters (plus and/or minus) with the following meaning:

<table>
<thead>
<tr>
<th>why</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>don’t print first nor ending line</td>
</tr>
<tr>
<td>+-</td>
<td>print first line but don’t print ending line</td>
</tr>
<tr>
<td>-+</td>
<td>don’t print first line but print ending line</td>
</tr>
</tbody>
</table>
If the parameter ⟨from⟩ is empty (use {} notation) then the printing starts at the begin of file.
If the parameter ⟨to⟩ is empty, only one line is printed. If ⟨to⟩=\end, then printing stops at the end of file.
If the line without something ({} notation) then the printing stops at next line with ⟨to⟩ pattern or at the end of the file.
The \iffirst command remembers the name of the included file and the number of the last line which was read.
Next time you can use the command \inext{}. This command starts the searching of the ⟨from⟩ pattern from the first line which wasn’t read by the previous \iffirst or \inext command. The parameters of the \inext command have the same meaning as the parameters of the \iffirst command. The parameter ⟨file⟩ is missing because the ⟨file⟩ from the last \iffirst command is used.
The number of the last line read by \iffirst or \inext command is stored in \lineno register (no matter if this line was printed or no). If the printing of code was stopped at the end of the file then \lineno equals to the number of lines of the file. You can do test of reaching of the end of file by \ifeof\infile.
Examples:
\iffirst {file.txt}{foo}{foo}{++} % print the first line
\iffirst {file.c}{//: from}{//:}{--} % the same as \ins command
\iffirst {file.h}{func}{}{++} % print of function prototype
\iffirst {file.c}{//\brace}{}{++} % print of the code block
\iffirst {file.c}{}{\empty}{}{++} % print of the first block
\iffirst {file.txt}{}{\empty}{--} % separated by empty line

If the first line of the code to be printed is empty then it is not printed. If the last line of the code to be printed is empty, it is not printed too. This is an implicit behavior. But if you write \skippingfalse, then this behavior is switched off. It means that the empty lines can occur at the begin or at the end of listings. You can use \skippingtrue in order to return to the implicit behavior.

The parameter ⟨from⟩ and ⟨to⟩ can have the prefix in the form {\count=⟨number⟩}. The value of the ⟨number⟩-th occurrence of the pattern have to be skipped and ignored during searching. The ⟨number⟩-th occurrence of the pattern is only significant. For example {\count=3,foo} means that two occurrences of foo have to be skipped and the third occurrence points to the right place, where the printing of the code starts (or ends).
If the prefix \count=⟨number⟩ is missing then DocBy.TeX supposes that \count=1.

If the parameters ⟨from⟩ or ⟨to⟩ are empty and \count=⟨number⟩ is used then the space after ⟨number⟩ needn’t be written and the meaning is slightly different: If the ⟨from⟩ parameter is empty then \count means the number of line from where the printing is started. If the parameter ⟨to⟩ is empty then \count means the number of printed lines. The previous sentences are true for ⟨why⟩=++ and
for \texttt{skippingfalse}. If the \langle why \rangle parameter have different value and/or \texttt{skippingtrue} then you must add/subtract one or two to/from the line number/number of lines. Examples:

\begin{verbatim}
\texttt{skippingfalse}
\texttt{ifirst }\{\texttt{file.txt}\}\{\texttt{count}=20\}\{\texttt{count}=10\}\{++\} % print from line 20 to 29
\texttt{ifirst }\{\texttt{file.txt}\}\{\texttt{count}=2\}\empty\{---\} % print to the second empty line
\texttt{ifirst }\{\texttt{file.txt}\}\{\texttt{count}=50\}\{\texttt{end}\}\{++\} % print from 50th line to the end
\texttt{ifirst }\{\texttt{file.tex}\}\{\texttt{count}=5\}\texttt{\nb section}\{\texttt{count}=2\}\texttt{\nb section}\{---\}
% print fifth section from TeX source
\end{verbatim}

\section*{2.7 References to Line Numbers}

The command \texttt{\cite[]} expands to the number of the line in source code. How to declare the \langle label \rangle? You can do it by \texttt{\ilabel [\langle label \rangle]}\{\langle text \rangle\}. command used before the \texttt{\ifirst} or \texttt{\inext} command. You can write more \texttt{\ilabel} commands if you want to declare more \langle label \rangle's hidden in the following listing. The order of \texttt{\ilabel} commands is irrelevant.

If the couple \langle label \rangle – \langle text \rangle is declared by \texttt{\ilabel} then the \texttt{\ifirst} or \texttt{\inext} command recognizes the occurrence of the \langle text \rangle in the listing. The line number of the first occurrence of \langle text \rangle is connected to the \langle label \rangle, it means the \texttt{\cite} expands to such line number.

The \langle label \rangle have to be unambiguous in the whole document. The \texttt{\cite} reference works forward and backward (after second pass of \TeX run).

The table of couples \langle label \rangle – \langle text \rangle created by set of \texttt{\ilabel} commands is local. It means that it cooperate only with the first \texttt{\ifirst} or \texttt{\inext} command. These commands use this table and reset it to the null state. You have to create this table before next \texttt{\ifirst} command again.

DocBy.\TeX does not write any warning if a \langle text \rangle doesn’t occur in the listing. Of course, if you use the unconnected \langle label \rangle by \texttt{\cite} command then the warning is printed.

The following example uses the known file \texttt{foo.c} mentioned in the section 2.2.

\begin{verbatim}
\ilabel [myfunct] \{function (float)
\ilabel [returnx] \{pair.x\}
\ifirst \{foo.c\}\{}\{}\{++
\end{verbatim}

The declaration of my very special function is on the line\texttt{\cite[myfunct]}. There is very specific idea on the line\texttt{\cite[returnx]} where the input parameter is multiplied by two.

\section*{2.8 Verbatim Environment by \texttt{\begtt}/\texttt{\endtt} and by Quotes}

Verbatim displays of the code can be included to the documentation by \texttt{\begtt} and \texttt{\endtt} pair of commands. The material to be displayed is written between these commands. All lines are inserted without changes, without interpretation of special \TeX characters. The lines are not numbered here and the occurrences of documented words are not hyperlinked automatically.

The following sections 3.2 and 4.8 discuss more possibilities of this environment.

You can write verbatim text in paragraph between quotes "...". This text is written by typewriter font and documented words are hyperlinked automatically. We recommend to use this environment for all parts of documented code which is mentioned inside the paragraph. This is analogical to math environment separated by $...$.

\section*{2.9 The Declaration of the Documented Word}

You can use commands \texttt{\dg}, \texttt{\dgn}, \texttt{\dgh}, \texttt{\dl}, \texttt{\dln} or \texttt{\dlh} in order to declare the documented word. The semantic of these commands is explained below. The syntax of these commands are slightly special. The purpose is to minimize the work of the writer, so the braces \{{}\} are not used, parameters are separated by space for instance. All these commands have the same syntax thus the example below uses only \texttt{\dg} command.

The possibilities of the syntax follows:
In general: The optional [ can follow after \texttt{\textbackslash dg} command. The \texttt{(text)} separated by \texttt{]} is read in such case and subsequent optional space is moved to the end of the \texttt{(text)}. It means that \texttt{\dg \texttt{[(text)]\word}} is the same as \texttt{\dg \texttt{[text]}\word}. Next, the \texttt{(word)} is read. The \texttt{(word)} parameter cannot include the space, comma, period, colon or semicolon because these characters can be separator of the \texttt{(word)}. These punctuation characters are not part of the \texttt{(word)} but they are printed. It means that \texttt{\dg\word:} prints \texttt{\word:} to the output and sets the \texttt{\word} as a documented word. If the scanned \texttt{(word)} ends by brackets \texttt{[} then these brackets are removed from \texttt{(word)} parameter, they are not printed in the current place but they are printed in footnotes and in the index.

Attention: the space have to be written after comma, period, colon or semicolon separator. If the space does follow immediately then the scanning process works only if the text between comma-like separator and space does not contain active characters ("..." for example). If the first character after space is \texttt{'} (backward quote) then the space and this quote is not printed.

Examples: \texttt{\dg (word)}\texttt{{/}next-text-without-line-breaking} or: \texttt{\dg (word)}\texttt{\textbackslash"...".}

The commands \texttt{\dgh, \dg, \dl, \dli} with space as a separator doesn’t print this separator because they usually print nothing (see below).

Semantic: The \texttt{(word)} parameter is documented word. If this \texttt{(word)} occurs on the other place in the document between "..." or in code listing then it is hyperlinked automatically (blue color). The documented word is highlighted by red color in the place where the \texttt{\dg} command is used and the optional \texttt{(text)} or \texttt{[]} does not printed. This is the destination of all blue hyperlinks. The \texttt{(word)} is printed in footnote of the current page too including the optional \texttt{(text)} in and/or including the optional \texttt{[}. The list of pages where the word is used is printed here too. The same is printed in the index. The index is sorted alphabetically by the \texttt{(word)}, not by the optional \texttt{(text)}.

The \texttt{(word)} declared by \texttt{\dg} is declared globally. This place is a reference point for the whole document.

The \texttt{\dgh} works like \texttt{\dg} but the word is not printed in the place of \texttt{\dgh} (mnemonic: \texttt{\dg} hidden). But this place is still the destination of hyperlinks and the word occurs in the footnote and in the index.

The \texttt{\dg} command (mnemonic: \texttt{\dg} next) saves its parameters but prints nothing. The first occurrence of the \texttt{(word)} in the next listing will be treated as the \texttt{\dg} is written here.

The \texttt{\dl} declares \texttt{(word)} locally. If the short name \texttt{(word)} is used in the same name space then it is hyperlinked and pointed to the place where \texttt{\dl} is used. The name space is changed by \texttt{\module} command. It means that \texttt{(word)} is used locally in the module. The word declared by \texttt{\dl} lives in two variants: short name \texttt{“(word)”} and long name \texttt{(depends on the current name space, typically “(word) / (module name)”)}). The long name is accessible in the whole document.

The section 2.10 explains the name spaces in more detail.

Each word can be declared at least once in the document else the error is printed by DocBy.Tex on the terminal. In case of \texttt{\dl} the short name is irrelevant but the long name have to be unambiguous.

The \texttt{\dli} command is \texttt{\dl} hidden and the \texttt{\dli} means \texttt{\dl} next. They are similar as \texttt{\dgh} and \texttt{\dg}.

If somebody hate this complicated parameter scanning then he/she can use internal commands with three parameters in braces: \texttt{\aidg, \aidgh, \aidg, \aidl, \aidli, \aidln}. The usage of the parameters is: \texttt{\aidg((text)){(word)}{(brackets)}}. Of course, you can do more by these commands: you can declare \texttt{(word)} with spaces or another delimiters, you can write something different than \texttt{“(”} as \texttt{brackets} parameter.
2.10 Namespaces

The namespace is a rule by which the short name of documented word is transformed to long name when \texttt{\textbackslash dl} is used. You can set the namespace by the command \texttt{\textbackslash namespace}. If the command \texttt{\textbackslash dl(\textbackslash lword)} is used inside the \texttt{\textbackslash namespace \{\textbackslash pre-text}\#1\{(\textbackslash post-text)\}\textbackslash endnamespace} environment then the short name is \texttt{\textbackslash lword} and the long name is \texttt{(\textbackslash pre-text)\textbackslash lword\{(\textbackslash post-text)\}}. All occurrences of \texttt{\textbackslash lword} are transformed to the long name inside the namespace environment. Outside of this environment the occurrence of short name \texttt{\textbackslash lword} is treated as no \texttt{\textbackslash dl} command is used. For example each word declared as \texttt{\textbackslash dl(\textbackslash lword)} inside \texttt{\textbackslash namespace \{\#1/uff\}\textbackslash endnamespace} environment is transformed to the long name "\texttt{\textbackslash lword}//uff" and the occurrences of \texttt{\textbackslash lword} inside this environment is hyperlinked and pointed to the place where \texttt{\textbackslash dl(\textbackslash lword)} is used. Outside of this environment only sequences \texttt{\textbackslash lword}//uff are hyperlinked.

The namespace is undefined out of \texttt{\textbackslash namespace...\textbackslash endnamespace} environment thus the \texttt{\textbackslash dl} command cannot be used here. The \texttt{\textbackslash module} command declares namespace \#1./\texttt{\textbackslash module(name)} thus you can use \texttt{\textbackslash dl} command for local functions and variables used in current module.

The long names are printed in the footnotes and in the index. The index is sorted by the long names alphabetically. The table of contents uses short names.

An example about namespaces follows:

```latex
\namespace {ju::#1} \%% namespace "ju" is set
The \texttt{\textbackslash dl aha} is declared here.
The word "aha" is hyperlinked to the place of its declaration.
The word "ju::aha" is hyperlinked too.
\endnamespace
\namespace {wow::#1} \%% namespace "wow" is set
The \texttt{\textbackslash dl aha} is declared here again.
The word "aha" points to the declaration inside "wow".
\endnamespace
\namespace \%% namespace off
The word "aha" is inactive here but the words "ju::aha" and "wow::aha" points to the right places.
```

The \texttt{\textbackslash namespace...\textbackslash endnamespace} environments can be nested. The inner environment have to have another namespace than the outside environment. These environments work globally independent of the \texttt{\textbackslash begin{vgrup}} and \texttt{\textbackslash end{vgrup}}. The \texttt{\textbackslash endnamespace} command used outside of all namespace environments does nothing. You needn’t to close these environments before \texttt{\textbackslash bye} command.

2.11 The Application Level of the Documentation

You can write the documentation to users of your code. For example the rules of the usage of functions are documented here (API) without codes of these functions. Suppose that you want to document the “inside behavior” of these functions by presenting their codes in the same document. The documented \texttt{\textbackslash lword} (a function name) can point to two different places in your documentation in such case: API documentation and CODE documentation.

The place with the function code (detail documentation) is located by \texttt{\textbackslash dg} command (or similar). The second place where the word is documented only for users without code can be declared by \texttt{\textbackslash api{\textbackslash lword}}. This command inserts the invisible mark only, the destination of links. The table of contents mentions the word and points to this place. The list of pages with the occurrences of the word (in the index and in footnotes) contains one underlined page number. This is the page where \texttt{\api{\textbackslash lword}} command is used. Of course, the \texttt{\api{\textbackslash lword}} command is not sufficient to including the word to the index. You need use the \texttt{\dg} command (in another place of the document) too.

The word declared by \texttt{\api} command are printed in the index with the \texttt{\apitext} prefix. The implicit value of \texttt{\apitext} macro is the special right arrow. You can see it in the index and in the table of contents in this document. The \texttt{\api\texttt{\textbackslash nb_api}} is used here but the code of \texttt{\api} macro is documented in section 5.9.

You can reference the place marked by \texttt{\api{\textbackslash lword}} by \texttt{\cite{\textbackslash lword}}. This macro expands to the page number where the \texttt{\api{\textbackslash lword}} is used. For example the \texttt{\cite{\nb_api\textbackslash lword}} expands to 11 in this document.

If there exist the API destination declared by \texttt{\api} command then the red word printed in the \texttt{\dg} place is hyperlinked and it points to the API destination. Typically, the occurrence of this word
is hyperlinked here with the \dg place as a destination. It means we have these two destinations cross hyperlinked.

2.12 Title, Parts, Sections, Subsections

Sections starts by \sec \langle Section Name \rangle \par command. Each section can include subsections started by the command \subsec \langle Subsection Name \rangle \par. Of course, the \par separator can be replaced by empty line (see the example in section 2.1). Sections and subsections are numbered automatically.

One or more sections can form a “part” which is started by \part \langle Part Name \rangle \par command. Parts are labeled by letters A, B, C, ... and they are printed more noticeable in table of contents than sections. The \part command does not reset the section numbering. It means that sections are numbered from one in the whole document, no matter if the document is divided into parts.

The \module\langle modulename\rangle \par command creates a new section Module \langle modulename\rangle, creates namespace and includes the \langle modulename\rangle.d file. You can change this default behavior, see sections 3.1 and 3.3.

The \title(\langle Name \rangle) \par command prints the title of the document by huge font in rectangle. If the \projectversion macro is defined then it expands to the text printed in the right upper corner of the rectangle by small font. The word “version” precedes. If our project has no version then you can define (for example):

\def\projectversion{\the\day. \the\month. \the\year}

The \author(\langle text \rangle) \par command centers the \langle text \rangle i the line and prints it bold. The common meaning is name(s) of the author(s).

The headline is created at each page of the document with the current section (from left) and title of the document (from right). You can redefine the right headline by new definition of the \headtitle macro.

The optional parameter \langle label \rangle in square brackets can be used with \sec and \subsec commands. The parameters looks like: \sec [[\langle label \rangle]] \langle Section Name \rangle \par. If the \langle label \rangle parameter is used then you can reference this place by \cite[\langle label \rangle]. This macro prints the number of referenced (sub)section and acts like hyperlink.

You can disable the transport of \langle (Sub)Section Name \rangle into table of contents by \savetocfalse command. Each section can include subsections \sec or \subsec command. This section has no number. The macro \emptynumber expands instead of number printing. This macro is set to empty by default. The \savetocfalse command influences only first \sec or \subsec command.

2.13 Hyperlinks, References

The destination of the hyperlink and/or reference have to be marked by \langle label \rangle. This can be done by optional parameter of the \sec or \subsec command (see the section 2.12) or by the command \label[\langle label \rangle] itself. You can make labels to line numbers of inserted code too (see the section 2.7). All labels have to be unambiguous in whole document (independent of their type).

The command \pgref[\langle label \rangle] expands to the number of the page where the \langle label \rangle is. The command \numref[\langle label \rangle] expands to the result which depends on the type of the destination:

- sections number if the destination is the section
- the pair \langle secnumber \rangle.\langle subsecnumber \rangle if the destination is the subsection.
- the number of the line if the destination is the line in the printed code
- empty if the destination is marked by \label command.

Both macros \pgref and \numref expand to the texts mentioned above without any more processing. It means that the printed text is not treated as hyperlink.

You can use the command \ilink[\langle label \rangle] \langle text \rangle in order to create the hyperlink in PDF mode. This macro prints the \langle text \rangle in blue color and it is treated as hyperlink to the destination specified by \langle label \rangle. For example the command \cite[\langle label \rangle] does the same as \ilink[\langle label \rangle]\numref[\langle label \rangle]. The real macro \cite executes a test if the \numref[\langle label \rangle] is empty and prints the \pgref in such case.

If the \langle label \rangle is not declared then \pgref[\langle label \rangle] and \numref[\langle label \rangle] have no destination. The \pgref expands to the text -1000 and \numref is empty in such case. These macros work on expand
processor level thus no warning message is implemented here. On the other hand the \cite command implements warnings. See the code of \cite on the page 36 for more detail.

The \module \langle modulename \rangle \label command creates the section with the label “\textit{m:\langle modulename \rangle}”. You can reference it by:

\def\refmodule[\#1]{\ilink[m:\#1]{\tt\#1}}

The \refmodule{\langle modulename \rangle} defined in the example above prints \langle modulename \rangle and creates it as hyperlink. For example \refmodule{\[base\]} prints the word “\textit{base}” in blue typewriter font and creates it as the hyperlink to the begin of the section “Module base” if this section is created by \module_{\textit{base}} \label command.

The \dg, \dgn and \dgh commands perform the command \label[\langle \textit{word} \rangle] internally and the \dl, \dln and \dlh perform the command \label[\langle \textit{longname} \rangle] internally. The \langle \textit{longname} \rangle is the long name of the \langle \textit{word} \rangle in context of the current namespace. For example, you can reference these places by \link[\langle \textit{word} \rangle]{{The,\langle \textit{word} \rangle}_{\text{\textit{documented},\textit{on the page}}}}{\pgref[\langle \textit{word} \rangle]}.

The \api\langle \textit{word} \rangle command executes \label[+\langle \textit{word} \rangle] internally. It means that you can reference this place by \link[+\langle \textit{word} \rangle]{{API: \langle \textit{word} \rangle}} for instance.

No more automatic numbering is processed by DocBy.T\TeX. Only numbers of sections, subsections and line numbers of the printed code. If you want to create the numbers of figures, publications etc. Then you have to write your own macros. You can use the \labeltext{\{\langle \textit{label} \rangle\}}{\{\langle \textit{text} \rangle\}} command in such case. This macro expands it parameters immediately and inserts invisible hyperlink destination into typeset material in horizontal mode. Then macro \numref\{\langle \textit{label} \rangle\} expands to \langle \textit{text} \rangle in the next pass of the T\TeX run. Example: we define the macro \bib\{\langle \textit{label} \rangle\} which inserts the destination marked by the \langle \textit{label} \rangle. The hyperlink with the number of the book can be created by \cite[\textit{b:\langle \textit{label} \rangle}].

\newcount\bibnum
\def\bib[\#1]{\par\advance\bibnum by1 \indent \llap{\the\bibnum} \labeltext{\[b:\#1\]}{\{\the\bibnum\}}\ignorespaces}

2.14 Pictures Inserting

The command \fig\langle \langle \textit{width} \rangle\rangle_{\langle \textit{picname} \rangle} inserts the picture into your document. The picture have to be prepared in the file \textit{fig/\langle \textit{picname} \rangle}.\textit{eps} (if DVI mode is used) and in the file \textit{fig/\langle \textit{picname} \rangle}.\textit{pdf} (if PDF mode is used). You can use another directory for pictures than \textit{fig/} – this name is stored in the \textit{\figdir} macro and you can redefine it. The \langle \textit{width} \rangle parameter is the ratio of the width of inserted picture to the \textit{\haize} (unit-less). The inserted picture is placed to left side with the paragraph indentation. For example \fig\langle 0.5 \rangle_{\langle \textit{foo} \rangle} inserts the picture from \textit{foo.pdf} (in PDF mode). The picture is scaled that its width is one half of the width of the printed text.

If you have the picture in \textit{eps} format and you need to convert it to \textit{pdf} then you can use:

\texttt{ps2pdf -dEPSCrop \langle \textit{picname} \rangle.\textit{eps}}

2.15 Items

The list of items are surrounded by \begitems and \enditems commands. The text is printed with one indent space (\parindent) more in this environment. These environments can be nested. Each item is started by \item \langle \textit{mark} \rangle. The \langle \textit{mark} \rangle is printed left from the text. If the \langle \textit{mark} \rangle is the star (*) then it is changed to the bullet. You can write \item \\textit{the\langle \textit{itemno} \rangle} if you want to print numbered items. The \textit{\itemno} register counts from one in each \begitems...\enditems environment.

The \item macro is redefined only inside \begitems...\enditems environment. If you wish to use the plain T\TeX macro \item then just don’t use \begitems and \enditems commands.

3 For Advanced Users

The definitions of basis macros of DocBy.T\TeX are mentioned in this section. The user can change these definitions if he need different behavior of DocBy.T\TeX than default one. For example, user documents different language than C and he/she redefine the \docsuffix macro or he/she redefine the code of \module and \ins commands completely.
3 For Advanced Users

3.1 Internal Names

The \doindex command creates new section with the name “Index”. The sections with names “Table Of Contents” or “Module” are inserted when table of contents is generated or \module command is executed. The word “version” is prefixed when the number of version is printed (if \projectversion is used). The text >>\PART is inserted into bookmarks by \part command. These texts are defined in the following macros: \titindex, \tittoc, \titmodule, \titversion and \opartname.

Note that different names are used by default when plain or csplain format is processed. But user can redefine these macros independently of the used format.

3.2 Hooks

Some more elaborate macros (\begtt, quotes, \ifirst, \inext, \doindex, \dotoc) execute so called “hook” before processing of more code. These hooks are macros and they are empty by default.

The \begtt and \endtt are processed. The \quotehook macro is inserted after begin of the group and after all catcodes are set by default before the text inside \begtt...\endtt is processed. The \indexhook macro is inserted by \doindex command after new section name is printed and before two column printing is activated. You can insert the notice to index here (see the index of this document for example). The \tochook macro is inserted by \dotoc command after new section name is inserted and before first line of table of contents is printed. The \bookmarkshook macro is inserted inside the group at the begin of bookmarks processing. You can set the different expansion of macros used in bookmarks here. For example \def\mylogo{MyProgram(R)}. Moreover, if you say \let\cnvbookmark=\lowercase here then all characters is converted to lower case in bookmarks. This is done by \lowercase primitive thus the different meaning of special characters can be set by \lccode. I use it for removing of accents because accents in bookmarks are interpreted by most PDF viewers wrongly. The \outputhook macro is inserted at the begin of the output routine. We recommend to set chosen macros to \relax meaning in order to they are not expanded in .ref file.

Examples:

```
\def\quotehook{\obeyspaces}  % normal spaces inside "...
\def\quotehook{\active}    % <text> is changed to \active
\def\begtt\{umbytein=1\} % auto-hyperlinks between \begtt...\endtt
\def\begtt\{setsmallprinting\} % \begtt...\endtt printed by small font
\def\begtt\{catcode\!'=0\} % !commands can be used in \begtt...\endtt
\def\indexhook{The special index with such and such properties follows.}
```
3.3 The Commands \module and \ins

The user documentation of these commands is in section 2.1. The \module \langle file \rangle command reads the file with the name \langle file \rangle docsuffix where docsuffix macro includes the suffix including the period.

\docby.tex
50: \def\docsuffix{.d} % implicit filename extension (for \module command)
51: \def\module#1{
52: \endnamespace\namespace{##1./#1}
53: \sec[m:#1]
54: \titmodule space #1 \par
55: \def\modulename{#1} \input #1 \docsuffix \relax
56: }

The \module command inserts the name of the file (without the suffix) into the auxiliary macro \modulename. This macro is used by the \ins \langle extension \rangle \langle text \rangle command.

\docby.tex
57: \def\ins #1 #2 {\ifirst{\modulename.#1}{//: #2}{//:}{--}}

3.4 The Comments Turned to Green Color

The \ifirst and \inext commands recognise C comments in the form //..\langle eol \rangle and /*..*/. These comments are printed in green color. You can disable this behavior by \noactive \langle string \rangle command. You can set a new type of comments by \setlinecomment \langle string \rangle command. These commands will be turned to green color from \langle string \rangle to end of line. These commands work globally. For example

\docby.tex
58: \ifx\mubyte\undefined
59: \def\setlinecomment#1{ }
60: \def\setlrcomment#1#2{ }
61: \else
62: \def\setlinecomment#1{\mubyte\linecomment##0 #1\endmubyte}
63: \def\setlrcomment#1#2{\mubyte\leftcomment##0 #1\endmubyte \mubyte\rightcomment#2\endmubyte \gdef\rightcomment{#2\returntoBlack}}
64: \if\mubyte\undefined
65: \fi

These macros are empty in no-enc mode. When enc\TeX{} is detected, they write information to enc\TeX{} table by \mubyte...\endmubyte primitive commands.

The \linecomment a \leftcomment commands are inserted by enc\TeX{} before each occurrence of declared character sequence. These commands sets the current color to green:

\docby.tex
67: \def\linecomment{\let\Black=\Green \Green}
68: \def\leftcomment{\global\let\Black=\Green \Green}

On the other hand, the \rightcomment command have to switch off the green color after the declared sequence is detected. Thus enc\TeX{} cancels the detected sequence and \rightcomment command returns this sequence back. After the returned sequence the \returntoBlack command set the current color to black.

\docby.tex
69: \def\returntoBlack{\global\let\Black=\oriBlack \Black}

Each line of listing is started by \Black switch. So, the green comments to the end of line work. But the green comment can be interrupted by the pair \Blue...\Black (see line 50 in previous section). In this case the \Black command have the \Green meaning so it returns to the green color. Next line is started with original \Black switch because each line is printed inside its own \TeX{} group.
The comments of type /*...*/ can affect more lines. So more lines have to be green and we re-define \Black to \Green globally. The lines starts with \Black command with \Green meaning in such case. The \returntoBlack returns to the original \Black switch globally.

DocBy.TEX initializes the comments by the rules of C language:

\begin{verbatim}
  72: \setlinecomment(/\/) \setlrcmmt(/\)(/\)
\end{verbatim}

4 For Designers

The documentation of macros which influence the look of the document follows. You can redefine it in order to change the design of your document. I mean that it is better to write simply and good documented macros for one purpose than the complicated macros with many parameters. You can simply use them or redefine them.

The main processing of docbytex is hidden in more complicated macros described in section 5. This differentiation of levels gives possibility to the designers to concentrate to design-like problems and not to drown in complicated recursive loops etc. of internal macros.

There are two different version of design macros: for pdfTEX mode and for DVI mode (without pdfTEX). This is the reason why you can see that the listings of following macros are often started by the text \ifx\pdfoutput\undefined.

4.1 Parameters and Auxiliary Macros

The parameters \hsize and \vsize are unchanged in DocBy.TEX. User can set his/her own preferred values. If they are unchanged by user then the default values from plain (usable for letter format) or csplain (usable for A4) are used.

DocBy.TEX sets new value for \parindent because we need more space here for colourised squares in section names.

\begin{verbatim}
  76: \parindent=30pt
\end{verbatim}

The \nwidth dimen is used like “narrowed \hsize” for many situations: the width of headline, footline and for title text.

\begin{verbatim}
  78: \newdimen\nwidth \nwidth=\hsize \advance\nwidth by-2\parindent
\end{verbatim}

The glue at the bottom of each page is set by \raggedbottom macro (defined in plainTEX). Moreover, the \exhyphenpenalty=10000 is set in order to deny the linebreaking after dashes (like pp 11–13).

\begin{verbatim}
  80: \raggedbottom
  81: \exhyphenpenalty=10000
\end{verbatim}

The fonts \bbf, \bbbf, \btt, \ttsmall, \rmsmall, \itsmall and \partfont are loaded here.

\begin{verbatim}
  83: \font\bbf=csb10 at12pt
  84: \font\bbbf=csb10 at14.4pt
  85: \font\btt=csst12
  86: \font\ttsmall=csst8
  87: \font\rmsmall=csr8
  88: \font\itsmall=csit8
  89: \font\partfont=csb10 at80pt
\end{verbatim}

The \setsmallprinting macro sets the typewriter font and prepares the \ttstrut of appropriate size and activates the line printing without vertical spaces between them by \offinterlineskip macro. The \parskip value is set to -1pt in order to a small overlaps of struts guarantee that no dashes-artsifacts occur at background of listings. The \setnormalprinting is similar.

\begin{verbatim}
\hsize: 13, 16, 22, 33, 39, 43 \vsize: 33, 43–44 \nwidth: 16, 19, 33 \bbf: 16, 18
\bbbf: 16, 18–19, 21 \btt: 16, 18 \ttsmall: 16–17, 20–22, 33 \rmsmall: 16–17, 19–20, 33
\itsmall: 16–17 \partfont: 16, 18 \setsmallprinting: 14, 17, 21–22 \ttstrut: 17, 21–22
\setnormalprinting: 17, 22
\end{verbatim}
The design is projected only with the following colors: \Blue, \Red, \Brown, \Green, \Yellow and \Black. If you need other colors you can define more similar macros.

All colors are defined by \setcmykcolor macro which is empty in DVI mode but a proper \special{PDF:\#1 k} is used in PDF\TeX{} mode. It means that the commands \Brown etc. can be used in DVI mode too, but they do nothing in that mode. The \oriBlack macro switches to black color and this macro is never changed. On the other hand, the \Black macro can be redefined in special environments and we need to return to real black color by \oriBlack macro at the end of such environment.

The \rectangle{⟨height⟩}{⟨depth⟩}{⟨width⟩}{⟨contents⟩} command creates a rectangle with specified dimensions and contents. This rectangle is filled by yellow color in PDF mode. The same rectangle has only black outline in DVI mode. Attention: the ⟨contents⟩ have to be prefixed by color switch otherwise it is invisible in PDF version (yellow on yellow). The \rectangle macro returns back to black color after rectangle is created.

The DocBy\TeX{} logo is typeset by \docbytext macro.

### Sections and Subsections

The \printsec{⟨sec-title⟩} and \printsecbelow macros are invoked from \sec macro. Their main task is to print the title of the section. You can redefine these implicit macros. You can concern with design of section here and you need not solve other problems (reference to the TOC, numbers, running heads etc.) which are hidden in \sec macro.

The following rules are mandatory: The vertical mode have to be initialized at the begin of the \printsec macro. Then you can insert vertical space and then you can insert the text of title. The \makelinks macro have to be inserted in the horizontal mode here. It creates the aim of hyperlinks. The \par command have to be the last command of your \printsec macro. No more vertical spaces

\Blue: 15, 17, 20, 36  \Red: 17, 20, 37  \Brown: 17–19, 21–22  \Green: 15–17, 36  \\
\Yellow: 17, 19–20, 22  \Black: 15–20, 22, 27, 33, 36–37  \setcmykcolor: 17  \\
\oriBlack: 15, 17, 20, 27  \rectangle: 17–21  \docbytext: 17, 34, 40  \printsec: 17–18, 34–35  \\
\printsecbelow: 18, 34–35

17
can be inserted here. The main \sec macro inserts another elements below the text and then it call the second macro \printsecbelow. The vertical space below the text is inserted from this macro (probably protected by \noerak). The right order of elements in \TeX’s vertical list is: “box, (whatsit, mark, etc.), penalty, glue”. The objects mentioned in the brace here is inserted by \sec macro. You can insert the “box” (by \printsec macro) and the “penalty+glue” (by \printsecbelow macro).

There are numerical registers \secnum and \subsecnum which store the actual (sub)section number. Moreover you can use the \ifsavetoc test. This is true if the title is printed in table of contents. If it is false then you can use \emptynumber macro instead of \the\secnum.

The \seclabel includes the \langle\!label\rangle of processed section or it is empty. You can use it for draft printing is you wish to see the labels (in margins, for example). DocBy.\TeX doesn’t implement this feature by default.

The \printsubsec and \printsubsecbelow macros does the same things but subsection is printed. They are invoked by \subsec macro.

The \printpart macro prints the title of part which is enumerated by uppercase letters. The \printpartbelow macro inserts the vertical space below the part title.

The \emptynumber is normally used if \savetocfalse. It prints nothing by default.

\title \langle\!title\rangle \par macro reads its parameter \langle\!title\rangle by auxiliary macro \secparam which ignores the possible space at the end of this parameter. This parameter is stored into \sectitle tokenlist and internal macro \iititle is invoked. This macro works in two different modes (DVI and PDF). The \langle\!title\rangle is stored into \headtitle macro (in both modes) only if the \headtitle is empty, it means that it not initialized by user. Then \iititle suppresses the headline printing on the current page by the \nohedaline command.
The \iititle macro expands to normal \centerline in DVI mode. On the other hand it creates the yellow rectangle of the width \nwidth in PDF mode.

If the \projectversion macro is undefined then its default value is empty.

The \author (author)\par does the same in both modes: prints the ⟨author⟩ text on the center by boldface font.

4.4 Headers and Footers

DocBy.TeX doesn’t change the output routine defined by plain TeX. It uses the standard plain TeX’s macros \headline and \footline when the design of headers and footers need to be changed.

The default design doesn’t do any difference between left page and right page because we suppose that the document will be read on monitor and may be printed without duplex.

The \footline prints the page number on center with \rectangle.

The text of \headline is changed during document is processed. It includes only \normalhead macro by default but if the \noheadline command is used then \headline changes its content until one page is printed.

The \normalhead macro stores page link by \savepglink and creates the header by nested \vbox/\hboxes. The name of section (\firstmark) is printed from the left side and the constant \headtitle is printed on the right side.

The \noheadline macro sets \headline to the temporary macro text which stores page link and does the change of \headline to its default value. This setting is global because we are inside the output routine.

The \headline macro prints the text in right side of header. It is empty by default but it is changed by \title command to the name of the document. User can define its value manually.

The auxiliary macro \headlinebox prints the empty rectangle in DVI mode and solid yellow rectangle in PDF mode. It is used on the line 188 for creating of square filled line in the header.
4.5 Printing of the Hyperlink Destinations and Footnote References

The hyperlink destination created by \dg or \dl macros are printed highlighted in order to reader can easy find it. The printing is processed by the macro \printdg{⟨text⟩}{⟨word⟩}{⟨brackets⟩} where the parameters are the same as in \iidg macro described in 2.9 section.

Only one parameter ⟨word⟩ is printed by default. The ⟨word⟩ is printed in rectangle in DVI mode or it is printed in red on solid yellow rectangle in PDF mode.

The red text is printed by auxiliary macro \printdgsinside. This macro prints only in red color if does not exist the \api destination. On the other hand it prints in red by \ilink macro if the \api destination does exist.

One item below the footnote rule is printed by \printfnote{⟨text⟩}{⟨word⟩}{⟨brackets⟩} macro (the parameters from \iidg macro are here). The ⟨word⟩ is printed in red, other information is printed in black.

The \specfootnote{⟨text⟩} macro is used here. It sends the ⟨text⟩ to the special footnote. The \pgref{⟨word⟩} returns the page number where the \api destination of the ⟨word⟩ is or it returns −1000 if \api destination does not exist. This number is stored in \apinum and if it is non-negative number then it is printed as first page number underlined. The list of page numbers where the ⟨word⟩ occurs is printed by \listofpages{⟨word⟩} macro. This macro ignores the number of page where \api destination is. The empty list of page numbers is detected by zero width of \box0.

4.6 The Index and Table of Contents Item

The \ptocline{⟨number⟩}{⟨text⟩}{⟨pageno⟩} command prints the item about a section or a part in table of contents. The \ptocsubline{⟨number⟩}{⟨text⟩}{⟨pageno⟩} does the same with the item about subsection. There is no substantial differences between these commands in DocBy.T\TeX’s default design, only one \indent more in \ptocsubline:
The `\mydotfill` command prints the dots in table of contents so they are aligned.

The `\ptocentry` \{\word\} \{(s-word)\} prints one item about documented word in table of contents. If it is `\api` occurrence of the \word then \{\type\} after \{(s-word)\} empty but if the \word is declared by `\d1` then \{(s-word)\} includes a short variant of the word and \word includes a long variant of it. We use long variant for hyperlinking and short variant for printing.

If someone want to print \text before \word or \text braces after \word then he can use a control sequence `\csname~\word\endcsname`. The example follows in the next macro \textbf{printindexentry}.

The `\myldots` command creates three dots, they are aligned with another dots in table of contents.

4.7 The Source Code Listing

The `\if` and `\next` macros print the required part of source code. They start with `\begingroup` and calls the `\printiabove` macro. Each line is printed by \texttt{\linestart \number} \{(text)\} \{\text\} macro. They finish by calling of `\printibelow` macro and `\egroup` command at the end. The designer can define these three macros. The default design makes differences between DVI and PDF mode.
The line above with file name is printed in DVI mode by \leaders primitive and \specrule macro. The line below listing is simple. In the PDF mode, we set \setsmallprinting at the start of listing and insert a small vertical space. The \printline macro sets the horizontal mode and strut is inserted here (in DVI mode) followed by box with number of the line. The interline penalty is 11 in the listing. In PDF mode, the solid yellow rectangle is printed by \rlap. We need to print the filename above the listing after the yellow rectangle of the first line is printed. That is the reason why there is the test if first line of the listing is printed by \isnameprinted control sequence. It is undefined by default but if the filename is printed then \isnameprinted is set to \relax (see lines 274 and 275). After the \egroup (inserted at the end of \first or \inext) the default value of \isnameprinted is restored. This value is undefined.

4.9 Pictures

The pictures are inserted in order to align their left side with the paragraph indent. The implicit design sets the \parindent to sufficient big value that the result is quite good. The width of the picture \figwidth is calculated as \hsize minus \parindent.

DVI mode: The macro \ifig⟨width ratio⟩,⟨filename⟩ inserts the picture from ⟨filename⟩.eps using the epsf.tex macro package. PDF mode: The macro inserts the picture from ⟨filename⟩.pdf by pdfTeX primitive commands \pdfximage, \pdfrefximage, \pdflastximage.
The \figdir includes the directory with the pictures.

4.10 Items

The macros for items mentioned in text are simple. The \begitems macro starts the items environment and the \enditems ends it. The \itemno register counts the number of the current item and the \dbtitem \langle mark \rangle is the global variant of \item macro. The \item macro is the same as in plain\TeX by default but it changes its behavior inside \begitems...\enditems environment.

5 For \TeX Wizards

The implementation of DocBy.\TeX is documented here. All internal macros of DocBy.\TeX are listed and commented in this section. May be, it is not so good idea to redefine these macros unless the reader want to do his own DocBy.\TeX.

5.1 Auxiliary Macros

The \dbtwarning macro prints warning on the terminal:

\begin{verbatim}
dbtfamily{\texttt{\textbackslash defsec \{ text \}, \texttt{\textbackslash edfsec \{ text \}} and \texttt{\textbackslash undef \{ text \}} define control sequence \texttt{\textbackslash csname \{ text \}}\texttt{\textbackslash endcsname}.}
\end{verbatim}

You can use the \texttt{\textbackslash undef} macro in following way:

\begin{verbatim}
\texttt{\textbackslash undef\{ text \}\textbackslash iftrue \langle sequence is undefined \rangle \textbackslash else \langle sequence is defined \rangle \textbackslash fi}
\end{verbatim}

You have to write \texttt{\textbackslash iftrue} after \texttt{\textbackslash undef\{ text \}}. There is a practical reason of this concept: you can use \texttt{\textbackslash undef test} nested inside another \texttt{\textbackslash if...\textbackslash fi} conditional.

The \texttt{\textbackslash nb} macro expands to normal backslash of catcode 12. You can use it if you need to search text with this character. The active tabulator is defined as eight spaces and auxiliary macros \texttt{\textbackslash obrace}, \texttt{\textbackslash cbrace}, \texttt{\textbackslash percent}, \texttt{\textbackslash inchquote} are defined here.

\begin{verbatim}
\texttt{\textbackslash obrace}: 8, 11, 15, 23, 30, 34, 36, 39–40 \texttt{\textbackslash cbrace}: 8, 23 \texttt{\textbackslash percent}: 8, 15, 23 \texttt{\textbackslash inchquote}: 8, 23 \texttt{\textbackslash softinput}: 24
\end{verbatim}
The \texttt{setverb} macro sets the cactodes of all special characters to normal (12).

\begin{verbatim}
339: \def\setverb{\def\do##1{\catcode'##1=12}\dospecials}
\end{verbatim}

5.2 \textbf{Initialization}

DocBy.TEX prints on the terminal:

\begin{verbatim}
343: \immediate\write16{This is DocBy.TEX, version \texttt{\dbtversion}, modes:
344: \ifx\mubyte\undefined NO\fi enc+
345: \ifx\pdfoutput\undefined DVI\else PDF\fi+
346: \ifnum\language=0 ENG\else CS\fi}
\end{verbatim}

The \texttt{\dbtversion} macro expands to the version of the DocBy.TEX. It is defined at the begin of the file \texttt{docby.tex}. If new version is released then this definition will be changed.

\begin{verbatim}
4: \def\dbtversion{(May 2014) % version of DocBy.TEX}
\end{verbatim}

If (pdf)csplain is used then the UTF-8 input is activated by encTEX. Unfortunately this isn’t compatible with DocBy.TEX which uses encTEX by different way. We need to deactivate the UTF-8 encoding input. If you need to write something in different language than English you need to use the 8bit encoding (ISO-8859-2 is usable for Czech when (pdf)csplain is used).

\begin{verbatim}
20: \input utf8off \csname clearmubyte\endcsname
\end{verbatim}

The encTEX mode is detected and initialized:

\begin{verbatim}
350: \if\mubyte\undefined % encTeX ??
351: \dbtwarning{encTeX is not detected}
352: \message{\space The documented words will be not recognized in source code.}
353: \message{\space Use pdftex -ini -enc format.ini to make
354: your format with encTeX support.}
355: \csname newcount\endcsname \mubytein
356: \def\enctextable@#1#2{\\\n357: \def\noactive#1{}\\\n358: \else \\\n359: \def\enctextable@#1#2{\\
360: \def\tmp #1,#1,#2\end{\ifx````````````\end{\fi}\end{verbatim}

The \texttt{\enctextable \{ (word) \}\{ (macrobody) \}} command inserts new item into encTEX table with the key \langle word \rangle. If this key is found by encTEX then it is removed from input stream and replaced by the \langle word \rangle macro which expands to \langle macrobody \rangle. For example after \texttt{dg foo} the key \texttt{foo} is activated for encTEX by \texttt{\enctextable\{foo\}\{\sword\{foo\}\}} command. If the \texttt{foo} is found in the input stream then it is replaced by \texttt{\sword\{foo\}}.

The \texttt{\enctextable} doesn’t store the key to the encTEX table if it is included in the list of prohibited words stored in \texttt{\owordbuffer}. The words are separated by comma here. They are prohibited because of \texttt{onlyactive}. The \texttt{\enctextable} defines only the \langle word \rangle sequence in such situation.

The \texttt{\noactive \{ (text) \}} macro inserts the \langle text \rangle as a key in the encTEX table. This key is not removed from input but the \texttt{\emptysec} control sequence is inserted before it. EncTEX is not able to remove the key from its table, it is only able to rewrite the behavior of the transformation process if the key is found. If we need to deactivate some key by \texttt{\noactive} then we rewrite its behavior.
All occurrences of documented words \langle word \rangle is transformed to \textcolor{red}{\textbackslashword{\langle word \rangle}} by \texttt{encTeX}. The hyperlink is created by this macro:

\begin{verbatim}
\fkef\textcolor{red}{\textbackslashword{\langle word \rangle}}{\ilink[@#1]{#1}\write\reffile{\string\reffuseword(#1)\the\pageno}}
\end{verbatim}

The \texttt{\textbackslashonlyactive \{\langle before\rangle\}\{\langle word\rangle\}\{\langle post\rangle\}} command inserts the \langle word \rangle into the list of prohibited words \texttt{\textcolor{red}{\textbackslashword{\langle word\rangle}} (only if this word isn’t here already). EncTeX changes all occurrences of \langle before\rangle\langle word\rangle\langle post\rangle to \texttt{\textcolor{red}{\textbackslashword{\langle before\rangle}\{\langle word\rangle\}\{\langle post\rangle\}}}. Moreover, the \langle word \rangle is deactivated by \texttt{\textcolor{red}{\textbackslashnoactive}} (may be it was activated when \texttt{\textcolor{red}{\textbackslashreffile}} is read). The \texttt{\textcolor{red}{\textbackslashword{\langle before\rangle}\{\langle word\rangle\}\{\langle post\rangle\}}} command prints \langle before\rangle by normal font, then runs \texttt{\not\textcolor{red}{\textbackslashword{\langle word\rangle}}} if it is defined (else prints \langle word \rangle normally). Finally, it prints \langle post\rangle by normal font.

\begin{verbatim}
\fkef\texttt{\textcolor{red}{\textbackslashword{\langle before\rangle}\{\langle word\rangle\}\{\langle post\rangle\}}}\owordbuffer,()\end{verbatim}

The DVI/PDF mode is initialized here:

\begin{verbatim}
\fkef\texttt{\textcolor{red}{\textbackslashword{\langle before\rangle}\{\langle word\rangle\}\{\langle post\rangle\}}}\owordbuffer,.\end{verbatim}

\subsection{5.3 The \texttt{\textbackslashiffirst}, \texttt{\textbackslashinext}, \texttt{\textbackslashilabel} Macros}

The \texttt{\lineno} register is the number of the line, \texttt{\ttlineno} register is the number of the line in the \texttt{\begtt...\endtt} environment. We use \texttt{\textcolor{red}{\textbackslashifcontinue}} for loop controlling and \texttt{\textcolor{red}{\textbackslashifskipping}} for setting \texttt{\skippingfalse} and \texttt{\textcolor{red}{\textbackslashskippingtrue}}.

\begin{verbatim}
\fkef\texttt{\textcolor{red}{\textbackslashiffirst\{\langle filename\rangle\}\{\langle from\rangle\}\{\langle to\rangle\}\{\langle why\rangle\}}}\end{verbatim}

The \texttt{\textcolor{red}{\textbackslashiffirst}} \{\langle filename\rangle\}\{\langle from\rangle\}\{\langle to\rangle\}\{\langle why\rangle\}} command analyses its parameter \langle why \rangle by \texttt{\textcolor{red}{\textbackslashreadparamwhy}} and tries to open the file \langle filename \rangle for reading by \texttt{\openin} primitive. If it is unsuccessful then a warning is printed. Else the \langle filename \rangle is stored to \texttt{\textcolor{red}{\textbackslashinputfilename}} macro and other parameters are analyzed by \texttt{\textcolor{red}{\textbackslashscaniparam}}. The \langle from \rangle resp. \langle to \rangle parameter is stored to \texttt{\textcolor{red}{\textbackslashtmpa}} resp. \texttt{\textcolor{red}{\textbackslashtmpb}} macro. The \langle num \rangle parameter from \texttt{\textcolor{red}{\textbackslashcount=\langle num \rangle}} is stored to \texttt{\textcolor{red}{\textbackslashtmpa}} and \texttt{\textcolor{red}{\textbackslashtmpb}} macros. The command \texttt{\textcolor{red}{\textbackslashinsinternal}} is invoked with expanded parameters \langle from \rangle and \langle to \rangle. The expansion is done via \texttt{\textcolor{red}{\textbackslashedef}} macro \texttt{\textcolor{red}{\textbackslashact}}.
The `\input{from}` `(to)` `(why)` macro does the analogue work as the `\iffirst`. The only difference is that the `filename` is not open by `\openin`. We suppose that the file is opened already. We are not sure that this is true and we check it by test of contents of the `\inputfilename` macro.

When the parameters `\from` and `\to` are expanded then we want to suppress all expansions of macros automatically inserted by enc\TeX. This work is done by `\noswords` macro.

The `\readiparamwhy` reads + or - characters from `\why` parameter and stores them to `\startline` and `\stopline` control sequences.

The `\scaniparam` `(param)` `(out)` `(outnum)` reads `(param)` in the form `\count=\text`. It stores the `\text` to the `\out` control sequence and `\text` to the `\outnum` control sequence. The prefix `\count` is optional thus we need to do a little more work to scan the parameters. This work is realized by auxiliary macros `\scaniparamA`, `\scaniparamB`, `\scaniparamC`. If the prefix `\count` is missing then `\outnum` is one.

The main work (inserting of source code) is done by the macro `\insinternal` with parameters `{\from}` `{\to}`.
The \isinternal macro has two main loops. First one (from line 436 to 440) reads the lines from input source file (by the macro \readnewline). Each line is stored to \etext macro. This loop finds the occurrence of the ⟨from⟩ parameter and nothing is printed.

The second loop (lines from 453 to 458) reads lines from input source file and searches the occurrence of the ⟨to⟩ parameter. The lines are printed by the \printilineA macro.

The preliminary work is done before first loop is started: the catcode, fonts and \mubytein setting. The \testline macro is defined here with the ⟨from⟩ separator. We will test the existence of ⟨from⟩ parameter by it. More flexible definition of the \testline macro is used here because of special form of ⟨from⟩ parameter (see user documentation in the 2.6 section). The end of loop is controlled by the \ifcontinue condition. The \nocontinue command runs the \continuefalse but not always. If \count>1, it means \tempnum>1, then the command only decreases the \tempnum by 1.

\def\nocontinue\ifnum\tempnum<1 \continuefalse \fi

The similar preliminary work is done before second loop. The \testline macro is defined again with the ⟨to⟩ separator. The searching process is similar as in the first loop.

The \ifx+\astartline is a test if user want to print the first line. The \ifx+\astopline is a test if user want to print the last line.

The \ilabellist macro tests the occurrence of labels declared by the \ilabel command.

The macro \returninsinternal \{text\}{possible-fi\}{ignore\} is more tricky. It is inserted when the end of the source file is occurred. The macro leaves its loop by the ⟨ignore⟩ parameter which is separated by the \printibelow text. Thus the part of the \insinternal macro is skipped to the line 463. The inserted conditionals have to be closed properly: the fi's are inserted here from the second parameter. The first parameter ⟨text⟩ includes the warning text if the warning have to be printed to the log file. If the ⟨text⟩ parameter is empty, no warning is printed.

\def\returninsinternal\#1\#2\#3\printibelow{
\ifx\#1\else\dbtwarning\{#1 in file \inputfilename\}fi
\ifx\#2\else\global\let\Black=\oriBlack % pokud jsme skoncili vypis uvnitr komentare
\fi
\fi
\ifx\#3\relax\else \printiline\{lastline\}\relax\fi
}

The \readnewline is simple:

\def\readnewline\{\read\inputfile to \text \global\advance\lineno by 1 \relax
\noswords \def\etext\{\\text\})

We are working with the line of source file in two versions: noexpanded line in the \text macro and expanded line in the \etext macro. The \noswords macro before expanding of the line guarantees that the \etext does not include control sequences created by encTeX (we need not these sequences when we are testing the occurrence of ⟨from⟩ or ⟨to⟩ parameter). The noexpanded \text version of the line (including the encTeXs sequences) is used when the line is printed.
The more intelligence is implemented in the \printilineA macro: the empty lines are printed with delay if the nonempty line follows. We need it because the last empty line have to be unprinted if skippingtrue. The \lastline macro has three states: \empty (at the begin), \relax (after the line is printed), (line number) (if the previous line is empty).

The \ilabellist macro stores all declarations from \label [⟨label⟩]⟨text⟩ commands. The empty value of \ilabellist have to be set as default.

The \begtt and \endtt macros are described in “\TeX book inside out” (the book in Czech language) in pages 27–30. The \startverb macro reads the following text separated by the \endtt. This text is divided into lines by \^M character of catcode 12. The loop is started by the \runttloop macro and the text is separated into lines. Each line is processed by the \printvline macro. The \endttloop is performed at the end of the loop. The final work is done here (the \printvbelow macro and the end of the group) and the next token is scanned by \scannexttoken macro. If this token isn’t \par then the following text is prefixed by \noindent. It means that \begtt...\endtt is “inside” the paragraph.
5.5 The Namespaces

Each name space is connected to its own \namespacemacro. This is a macro with one parameter which is declared by \namespace\{⟨macro-body⟩\}. The \namespacemacro is empty by default.

We need to set a label to each name space. The label is the text expanded by \namespacemacro{⟨label⟩} and the mark ⟨nslabel⟩ is used for such label in this documentation. There is a little risk that the ⟨nslabel⟩ is ambiguous but I hope that this situation will not occur.

Each name space has to know all local words declared in it in order to have occurrence of this local word can be referenced to the \dl declaration; the \dl declaration can be used after first occurrence of such word. The encTeX tables have to be initialised with all local words at the start of the name space. The original state of these tables have to be restored at the end of the name space. We cannot wait to the \dl command but we need to use the \reffile file. It means that the name spaces are inactive in the first TeX’s run.

The macro \ns⟨nslabel⟩ includes the list of all locally declared words in the namespace ⟨nslabel⟩ after the \reffile file is read. The list has the following format:

\locword{⟨word1⟩}\locword{⟨word2⟩}\locword{⟨word3⟩}...

Because the encTeX table setting is global, we define all namespace macros globally too. This is the reason why \namespace...\endnamespace is independent of groups in TeX.

The \namespacemacro is defined at the start of the \namespace command. The original value of the \namespacemacro is stored to the \no:⟨nslabel⟩ macro in order we are able to restore this value at the end of the namespace...\endnamespace environment. Next we define the macro \locword so that the encTeX table is set after invoking of the \ns⟨nslabel⟩. The \locword macro stores the original meanings of redefined control sequences first.

The \endnamespace command redefines the \locword macro so that the original meaning of redefined sequences are restored. If the original meaning is “undefined” we need to store the \nword to the encTeX table in order there is no possibility to clear the item from encTeX table definitely. The original value of the \namespacemacro is restored by the \no:⟨nslabel⟩ macro.
These macros use the \texttt{\write} sequence which writes the text to the \texttt{\reffile} with delay (in output routine) but the \texttt{\nb} control sequence is not expanded. Enc\TeX\ stores the control sequence \texttt{\langle word\rangle} instead local \texttt{\langle word\rangle} at the start of each name space. The \texttt{\langle word\rangle} macro expands to \texttt{\langle word\rangle}. If the local word occurs the \texttt{\lword{⟨word⟩}} works as follows:

\begin{verbatim}
\def\write#1{\if\relax#1\relax\else\edef\act{\expandafter\noexpand\csname ii\endcsname}{#1}}
\edefsec{\string\namespacemacro@{}\endcsname}{\write{#1}}
\end{verbatim}

The \texttt{\genlongword} \langle temp \rangle \langle word \rangle command creates long version of the \langle word \rangle from short variant of it and stores this log version to \langle temp \rangle macro. The occurrence of the \langle word \rangle is presented by the parameter of the \texttt{\link} command and by the long name (unambiguous) written to the \texttt{\reffile}. The short variant of the word is printed.

The reading of the \texttt{\reffile} is controlled by \texttt{\refns \langle nslabel \rangle} macro. This control sequence is stored to the file at the begin of the name space. The second control sequence \texttt{\refsend \langle nslabel \rangle} is stored at the end. The items of the type \texttt{\reffid{⟨text⟩}{⟨long-word⟩}{⟨brackets⟩}{⟨short-word⟩}} are stored between these control sequences. We read only the items with the nonempty \langle short-word \rangle parameter. These items are stored by \texttt{\dA} command.

\begin{verbatim}
\def\refns#1{\if\relax#1\relax\else\edefsec{\string\namespacemacro@{}\endcsname}{\refns{#1}}\fi}
\def\refsend#1{\if\relax#1\relax\else\edefsec{\string\namespacemacro@{}\endcsname}{\refsend{#1}}\fi}
\end{verbatim}

The \texttt{\refns} macro remembers the previous \langle nslabel \rangle which is stored in \texttt{\currs}. This value is stored to the \langle o⟩: \langle new-nslabel \rangle and the \texttt{\currs} is redefined as \langle new-nslabel \rangle. The implicit value of the \langle ns: ⟨nslabel \rangle is empty. The \texttt{\reffid} commands add information to the \langle ns: ⟨nslabel \rangle buffer (see lines 849–852 in section 5.9). Finally, the \texttt{\refsend} command returns the \texttt{\currs} macro to the original value before name space was started.

### 5.6 The \texttt{\dg} Command and Friends

The macros \texttt{\dg, \dA, \dgN, \dgH, \dln, \dlh} save its name to a \texttt{\tmpA} and then they scan parameters by a \texttt{\dparam}. Finally they run the internal version for itself \texttt{\csname ii \endcsname \tmpA \endcsname}. The macros \texttt{\dgpar} save parameters by a \texttt{\dparam} and then they scan parameters by a \texttt{\dparam}. Finally they run the internal version for itself \texttt{\csname ii \endcsname \tmpA \endcsname}. The previous macros prepare the reading of optional parameter. The main work is done by the \texttt{\dparam} macro.

\begin{verbatim}
\def\dparam #1#2 { %
  \if\relax#1\relax#1\relax#2\relax\else\edef\act{\expandafter
\noexpand\csname ii\endcsname}{#1}{#2}\fi
\end{verbatim}
5 For TtX Wizards

\begin{verbatim}
568: \expandafter\act
569: \tmpb \if|\expandafter\ignoretwo\tmpa|\expandafter\maybespace|\fi
570: )
571: \def\nextdparam[#1]|\maybespace|\fi|\fi|\def\param[#1]{}

If there is a space after closed bracket } then the \#2 parameter is empty (it is separated by space). The \dparam macro runs again in such case (by \nextdparam macro which scans the rest of parameters of the \dparam). The space is inserted inside the braces before the \dparam is run again. Now, we can separate the \#2 parameter (it means the \langle word\rangle) to the part before the first comma or period or colon or semicolon and to the second part with the rest. The first part is stored to \tmpa and the second part (including the separator) is stored to \tmpb. This work is done by the macro \varparam:

\begin{verbatim}
572: \edef\varparam#1|\def\tmp[#1]|\#2|\#2|\if\#2\else\fi\end\#2|\fi
573: \expandafter\gobblelast\tmpb|\end\#2|\fi|\#2|\#2|\if\#2\else\fi\end\#2|\fi
574: \edef\gobblelast#1|\end\#2|\def\tmp#2|\#2|\def\#2|\#2|\end\#2|\fi
575: \expandafter\tmp
576: \def\gobblelast#1|\end\#2|\def\tmp#2|\#2|\def\#2|\#2|\end\#2|\fi
\end{verbatim}

The macro \varparam(separ) defines the temporary macro \tmp#1|\langle separ\rangle#2, which is run by \tmp\langle word\rangle|\langle separ\rangle\. If the \#2 is empty then the explicitly written \langle separ\rangle was used as separator and the \langle word\rangle does not include the \langle separ\rangle. The \tmp still includes the \langle word\rangle in such case. On the other hand, if the \langle word\rangle includes \langle separ\rangle then we need to store the rest after the \langle separ\rangle to \tmpb including such \langle separ\rangle. The \#2 parameter includes \langle rest\rangle|\langle separ\rangle. The desired work is done by the \gobblelast macro with the parameter \langle contents of \tmpb\rangle|\end\langle separ\rangle|\langle rest\rangle|\langle separ\rangle. The \#1 includes the \langle rest\rangle and the new \tmpb is filled up by \langle separ\rangle|\langle rest\rangle|\langle old contents of \tmpb\rangle.

At the end of this work, we have the \langle word\rangle in \tmpb but it can be followed by \$. This problem is solved by \managebrackets macro which separates these braces if they exist. The braces are stored to \printbrackets in such case.

\begin{verbatim}
578: \def\managebrackets#1|\#2|\end|\def\tmp#1|\if\#2\else\\\\\\\end\#2|\fi
579: \if\#2\else\\\\\\\end\#2|\\\\\\\end\#1|\\\\\\\end\#1|\fi
\end{verbatim}

The \maybespace macro prints the space after the contents of \tmpb only if the name of the macro used by the user has only two letters (\dg, \dl) and the character ‘ follows.

\begin{verbatim}
581: \def\maybespace|\futurelet\tmp|\domaybespace
582: \def|\domaybespace|\let|\next=|\space
583: \if\tmp|\tmpd|\dnext#1|\#1|\fi
584: \next
\end{verbatim}

The \dparam macro changes the original command \dg, \dl respectively to internal variant \idg, \iidl respectively. This is done on the line 567. Parameters are expanded before the internal macro is started. Now, we’ll concentrate to the internal macros.

The \iidg macro inserts the \sword to the encTtX table (this is redundant because the same work is done when \reffile is read by \reffdg macro). Next, the \iidg macro creates the aim of the reference in the form \#\langle word\rangle and saves \reffdg(\langle text\rangle\{\langle word\rangle\{\langle long-word\rangle\}\{\langle brackets\rangle\}) to the \reffile. The highlighted \langle word\rangle is printed by the \printdg command and the footnote is inserted by the \printfnote command.

\begin{verbatim}
586: \def\iidg#1|\#2|\#3|\if\#2\else\\end\#2|\fi
\end{verbatim}

The \iidl creates the aim of the reference by \label\#\langle long-word\rangle, writes the information to the \reffile in the format \reffdg(\langle text\rangle\{\langle long-word\rangle\{\langle brackets\rangle\}\{\langle short-word\rangle\}) and prints the \langle short-word\rangle highlighted by \printdg command. It stores nothing to the encTtX table. Finally, it inserts the footnote by \printfnote(\langle text\rangle\{\langle long-word\rangle\}{\langle brackets\rangle}).
The \iidgh a \iidlh macros do the same work as the non-h variants. The only difference is that they do not print the word. The \printdg is redefined locally in order to do nothing.

The \iidgn command redefines the \word macro which is inserted to the text by enc\TeX. The result of the expansion will be \sword\{\text\}{\word\}{\brace} instead of the common result \sword\{\word\}.

The tasks of the \fword macro are: do \iidgh, print the \word in red and return the \word macro to the normal state.

The \iidln macro stores the current meaning of the \word to the new control sequence \;\word and redefines the \word. The result of the expansion is \flword\{\text\}\{\word\}\{\brace\}. The tasks of the \flword macro are: do \iidlh, print \word in red, return the original meaning of the \word (from the \;\word storage). If the \;\word is undefined we need to inactivate the \word macro by \nword because there is no possibility to remove the item from enc\TeX table.

5.7 The Special Footnotes

The footnotes are placed beside each other. There are only words which are declared on this page by \dg. Because this concept is visual incompatible with the “normal” footnotes, we deny them:

Our special footnotes use the “insert” \footins declared in plain\TeX. The problem is to estimate the vertical space of one footnote when these footnotes are beside each other. The dirty trick from the \TeXbook (to insert the inserts by percent of the width) is not used here because the pagebreaks didn’t converge in the sequence of \TeX runs. The second run gets the pagernumber lists in the footnotes but they are not definite because of new pagebreaks. The new pagebreaks influence new lists of pagernumbers in footnotes and the new lists influences the new pagebreaks because the widths of the footnotes are different from previous \TeX run. The oscillation is very common in such case.

I decided to work only with the average space of the footnotes common for each of them. This coefficient is the number of the lines of the footnotes divided by the number of the footnotes. Each footnote inserts to the vertical list the space of the line height (10pt) multiplied by this coefficient. I need to set the \count\footins only.
In order to guarantee the convergence of this problem, we need to fix the coefficient (mentioned above) after second \TeX run. If this coefficient is changed in each \TeX run then the unconvergence is very possible. The value of this coefficient after first \TeX run is unusable because the lists of pagenumbers in footnotes are empty at this state. The implicit coefficient is set to \texttt{\count\footins=200} for first and second \TeX run (we suppose five footnotes on the one line).

The average coefficient (instead of the width of each footnote) can produce a little overfull or underfull pages. We need to have the resource for this situation in \texttt{\skip\footins} and we need to use the vertical glue above and below the footnote rule.

\begin{verbatim}
docby.tex 622: \skip\footins=18pt 623: \dimen\footins=\vsize 624: \count\footins=200
\end{verbatim}

The \texttt{\totalfoocount} accumulates the number of the footnotes and the \texttt{\totalfoodim} accumulates the total height of all lines with footnotes.

\begin{verbatim}
docby.tex 626: \newcount\totalfoocount
627: \newdimen\totalfoodim
\end{verbatim}

The \texttt{\specfootnote \{⟨text⟩\}} macro inserts to the \footins one \texttt{\hbox{⟨text⟩}} and advances \texttt{\totalfoocount} by one.

\begin{verbatim}
docby.tex 629: \def\specfootnote#1{\insert\footins\bgroup
630: \let\tt=\ttsmall \rmsmall
631: \floatingpenalty=20000 \setbox0=\hbox{#1}
632: \ht0=10pt \dp0=0pt \box0 \egroup
633: \global\advance\totalfoocount by1
634: }
\end{verbatim}

I decided to keep the output routine of plain\TeX unchanged. It means that the part of this routine which solves the footnote printing was needed to change. The \texttt{\footnoterule} macro of plain\TeX was redefined. The \texttt{\unvbox\footins} separator removes the same text from the original output routine.

\begin{verbatim}
docby.tex 636: \def\footnoterule \unvbox\footins { 637: \vskip-12pt \vfil 638: \moveright\parindent\vbox{\hsize=\nwidth \hrule
639: \setbox2=\vbox{\unvbox\footins \unskip
640: \setbox2=\lastbox
641: \global\setbox4=\hbox{\unhbox2}
642: \loop \unskip\unpenalty
643: \setbox2=\lastbox
644: \ifhbox2 \global\setbox4=
645: \hbox{\unhbox2 \penalty-300\hskip15pt plus5pt \unhbox4}
646: \repeat}
647: \setbox2=\vbox{\hrulebox} \parskip=0pt
648: \lineskiplimit=0pt \baselineskip=10pt \raggedright \rightskip=0pt plus7em
649: \leftskip=0pt \hyphenpenalty=10000 \noindent \Black \unhbox4 }
650: \global\advance\totalfoodim by\ht2 \unvbox2}
651: }
\end{verbatim}

This macro decomposes the vertical list of inserts \texttt{\footins} and composes them again beside each other in horizontal box 4. The raggedright parameters are set and the box 4 is unboxed in horizontal mode ended by \texttt{\endgraf}. This means that the footnotes are divided to lines. The \texttt{\totalfoodim} is advanced here too.

The \texttt{\bye} macro (see the line 789) writes the \texttt{\totalfoocount} and \texttt{\totalfoodim} to the \texttt{\reffile}. The actual average coefficient is added here too. This information is written only if the \texttt{\indexbuffer} is not empty, it means that (at least) the second \TeX run is in progress.

This information is read by the \texttt{\refcoef \{⟨coef⟩\}\{⟨number⟩\}\{⟨height⟩\}} macro at the start of the next \TeX run. It sets the average coefficient \texttt{\count\footins}. The change from implicit value 200 to the new value is done only once. Next \TeX runs keep this value unchanged. The auxiliary macro \texttt{\gobblerest} removes the digits after decimal point including the text pt.
5.8 Section, Subsection, Part

The \texttt{secnum}, \texttt{subsecnum}, \texttt{sectitle} and \texttt{ifsavetoc} are declared here. \texttt{savetoc} is true by default.

There is an optional parameter \texttt{[(label)]} followed by optional (ignored) space when \texttt{sec} and \texttt{subsec} macros are used. The last token of \texttt{(title)} can be space too and we need to ignore it. This is reason what the macros are somewhat complicated. The name of the macro is stored to \texttt{secparam} and the parameter scanning process is started by \texttt{tmpA}.

The \texttt{secparam} reads the optional \texttt{[(label)]}. If it exists then it is stored to \texttt{seclabel} macro else \texttt{seclabel} is empty. The \texttt{seclabel} macro ignores optional space after the \texttt{)}. The \texttt{secparamB} \texttt{(title)} \texttt{par} macro reads \texttt{(title)}. The unwanted space at the end of the \texttt{(title)} is removed by \texttt{nolastspace} macro which cooperates with the \texttt{setparamC} macro. This macro stores the \texttt{(title)} (without the last space) into \texttt{sectitle} and executes \texttt{iisec} or \texttt{iisubsec}.

The \texttt{iisec} macro sets the \texttt{secnum} and \texttt{subsecnum} values and defines \texttt{makelinks} where the hyperlinks are prepared (used by \texttt{printsec}). The \texttt{printsec} macro prints the title of the section. The information of the type \texttt{reftocline \{secnum\}\{(title)\}\{(pagenumber)\}} is stored to \texttt{reffile}. The command \texttt{mark\{(secnum)\\{(title)\}}} is executed and the vertical space is appended by \texttt{printsecbelow}.

\begin{verbatim}
\texttt{secnum}: 18, 34–36 \texttt{subsecnum}: 18, 34–35 \texttt{sectitle}: 18–19, 34–35 \texttt{ifsavetoc}: 18, 34–35
\texttt{sec}: 12, 15, 17–18, 34, 39 \texttt{subsec}: 12, 18, 34 \texttt{tmpA}: 18, 25–26, 30–31, 34–35
\texttt{secparam}: 18, 34–35 \texttt{seclabel}: 18, 34–35 \texttt{secparamB}: 34 \texttt{secparamC}: 34 \texttt{nolastspace}: 34
\texttt{setparamC}: iisec: 34–35 \texttt{makelinks}: 17–18, 35
\end{verbatim}
5.9 Links and References

The hyperlinks are solved by \savelink \{label\} and \iilink \{label\}\{text\} macros. The \savelink stores the invisible destination into document raised to the height of \linkskip above baseline. The \iilink (i.e. internal link) is documented in the 2.13 section. The \savepglink saves the numerical destination (page number) which will be used by \pglink if a page is referred.

\begin{verbatim}
736: \ifx\pdfoutput\undefined
737: \def\savelink{[]}
738: \def\iilink{\{label\}\{text\}}
739: \def\savepglink{}
740: \def\pglink{\afterassignment\dopglink\tempnum=}
741: \def\dopglink{\the\tempnum}
742: \def\unlink{\{\the\part\}}
743: \else
744: \def\savelink{[]}\ifnum\nointerlineskip\fi
\end{verbatim}
These macros have special implementation for DVI and PDF modes. The blue color for links is declared in the \link macro. You can change this feature by changing of this macro.

The internal labels for PDF links cannot include backslashes. That is the reason why the \nb (normal backslash) macro is redefined here. We expect the expanded parameter of \savelink and \ilink macros.

The \savepglink macro (see above) is used by \headline, this places the destination at every page. The \pglink \number macro reads the \number (it is numerical register or number itself) and creates the link to the \number. The \number is printed in blue color and it is clickable. The numerical register is scanned by \afterassignment followed by \dopglink.

The line \reflabel \{\label\}{\text}{\page} is stored to the label file by \jobname.ref file by \labeltext macro. This information is read by \reflabel macro and stored in ~X\{\label\} and ~Y\{\label\} control sequences. These sequences are used by \numref a \pgref. Note that if the \text is empty (this is a case of documented words for example) then the control sequence ~X\{\label\} is not defined. This saves the \TeX memory for names of control sequences.

The \labeltext \{\label\}{\text} stores the desired information as pronounced above. First, it creates PDF link by \savelink macro and second, it stores data to .ref file. The \writelabel \{\label\}{\text} we used for this purpose which expands to the asynchronous \write primitive (in order to save right value of the page number). We need to expand the \text parameter because \the\secnum (or similar data) is here. This is a reason why the parameters are switched (the \text parameter cannot be expanded) and the auxiliary macro \writelabelinternal \{\text\}{\label} is used. First part, i.e. \writelabel \{\text\} is expanded by \edef.

The \label is defined simply as “empty” \labeltext.

The \cite \{\label\} macro prints the hyperlink. The warning on the terminal is printed when \label is misspelled. The macro is documented in 2.13 section.
The links are solved in \api\{\langle word \rangle\} too. This macro uses \label[+\langle word \rangle] and saves the \langle word \rangle prefixed by \reffilerefapiword to \reffile.

The \apitext is printed alongside the \langle word \rangle in the table of contents and the index.

When the \bye is executed, the information for \reffile (line 789) is stored in \reffile and the test of \reffile data consistence is processed.

The test of \reffile data consistence is done by following steps. First the \reffile is closed, then the control sequences used in \reffile are redefined by \setrefchecking macro, then the \reffile is read again. Now the macros from \reffile do the test itself. If inconsistency occurs then the \continuefalse is executed. We can ask to the result of the test by \ifcontinue conditional. The elaborate check of all automatically generated hyperlinks is done after the \reffile is read. This check is realised by \indexbuffer. Why? See the \setrefchecking.

\api: 11, 13, 20–21, 37–38  \apitext: 11, 21, 37  \bye: 7, 11, 33, 37  \setrefchecking: 37
The \refcoef macro is redefined here: it does nothing. Next, the new version of the \reflabel checks if the reference is in the same page as in the last run and if it has the same text. The new macro \refuseword works as its original, only the \word control sequences are used instead \w:\word. These control sequences are used for another purpose than during normal processing. First, these sequences take the \relax meaning at line 803. Second, the \, is redefined in order to do the test of equivalence of the \word and \word sequences. The test is executed by \indexbuffer\relax at line 788. If an inconsistency occurs then the message is printed and macro processing is skipped to \ignoretorelax. next, the \refdg macro is redefined: it writes data only to \tocbuffer. The other macros from \reffile write data to \tocbuffer too. The old contents of \tocbuffer is stored to the \text and the new one is created during \reffile reading. We check if the table of contents is changed at line 794.

5 Generating of Table of Contents, Index and PDF Outlines

The table of contents (TOC) and index can be printed at various places in the document (at the begin, end, in the middle…). We need to print them correctly independent of their position. The \reffile can be read only at begin of the document. After that, it is cleared and reopen to write new information. So, we need to store all desired information for TOC or index printing during reading of the \reffile. We are using the \tocbuffer and \indexbuffer macros for this. First, these “buffers” must be set as empty. The \addtext ⟨text⟩\to(buffer) is used for adding new ⟨text⟩ to the ⟨buffer⟩.

The following commands are used in the \reffile.

\reftocline⟨number⟩⟨title⟩⟨page⟩ % about section, subsection for TOC
\refdg⟨before⟩⟨word⟩⟨after⟩⟨word⟩ % about usage of \dg, \dl
\refapiword⟨word⟩ % about usage of \api⟨word⟩
\refuseword⟨word⟩⟨page⟩ % about the existence of ⟨word
\reflabel⟨label⟩⟨text⟩⟨page⟩ % see section 5.9, links, references
\refcoef⟨coefficient⟩⟨number⟩⟨height⟩ % see section 5.7, spec. notes
\reftocline⟨number⟩⟨title⟩⟨pagenumber⟩ % about section, subsection for TOC.

The \reftocline ⟨number⟩⟨title⟩⟨pagenumber⟩ macro is used for TOC.

The information about all sections and subsections are stored in \tocbuffer gradually. This buffer includes control sequences \dotocline ⟨number⟩⟨title⟩⟨page⟩. The only difference between section and subsection is stored in the ⟨number⟩ parameter: subsection has the ⟨number⟩ with a period. This difference is recognised by the \istocsec macro.

The \tocbuffer includes TOC information about sections and subsections. Moreover, it includes the data about documented words stored by \refdg a \refapiword.
The `\refdg` macro has `{\langle before\rangle}{\langle word\rangle}{\langle after\rangle}{\langle k-word\rangle}` parameters where `{\langle before\rangle}` is a text before word, `{\langle word\rangle}` is a long variant of the word and `{\langle after\rangle}` can include optional braces `{}`. If the long word differ from short word (when `{\backslash dl}` is used) then `{\langle k-word\rangle}` includes the short variant of the word else `{\langle k-word\rangle}` is empty. The `\refdg` macro stores its information to `{\tocbuffer}` and `{\indexbuffer}` in parallel. If `{\langle k-word\rangle}` is empty then `{\word}` is stored to enc\TeX table. If `{\langle k-word\rangle}` is nonempty then namespaces are taken into account. The TOC is created by the `{\dotoc}` macro.

The Index is created by `{\indexbuffer}` which includes the list of all declared words in the document. Each word is stored in the form of control sequence (this takes minimum \TeX memory) and they are separated by comma (before sorting) or `,` (after sorting):

before sorting: \langle word1\rangle, \langle word2\rangle, \langle word3\rangle, \langle word4\rangle, ...
after sorting: \langle\langle wordA\rangle\rangle, \langle\langle wordB\rangle\rangle, \langle\langle wordC\rangle\rangle, \langle\langle wordD\rangle\rangle...

The \langle word\rangle means one control sequence here. Each control sequence is a macro with the body `{\langle before\rangle}{\backslash right}{\langle after\rangle}`, see the 844 line. The index is printed by the `{\doindex}` macro.

The `{\calculatedimone}` command and the test of `{\dimen1}` value prepares the two columns typesetting, see the section 5.13. The `{\doindex}` begins with `{\sec}` with the `{\titindex}` title. The index printing is started when `{\indexbuffer}` is nonempty. The `{\indexbuffer}` is sorted by `{\sortindex}` (see section 5.11). Then the two columns printing is opened by `{\begmultii2}` and the `,` separator takes the meaning `{\doindexentry}`. This macro prints each index entry when `{\indexbuffer}` expands.
The \texttt{\doctypeindexentry} macro removes the \texttt{-} characters from the control sequence \texttt{-⟨word⟩} by the \texttt{\ignoretwo}, so the \texttt{\tmp} includes \texttt{⟨word⟩} only. If the \texttt{⟨word⟩} begins by backslash, it is replaced by \texttt{\nb} using \texttt{\remakebackslash} macro. The reason: we needn’t the backslash in the PDF internal labels, see \texttt{\pdfoutput} for more information. The index entry is finally printed by the \texttt{\printindexentry} macro.

When the PDF outlines are created, we need to know the number of children of each node in the outlines tree. This number is calculated when \texttt{\reffile} is read by \texttt{\addbookmark ⟨node⟩} macro (see \texttt{\reftocline} and \texttt{\reffile} macros). The parameter \texttt{⟨node⟩} can be the number of section or the \texttt{⟨section⟩}, \texttt{⟨subsection⟩} pair. The \texttt{⟨node⟩} for which we are calculating children is saved in \texttt{\currbb} macro. The \texttt{\currssec} includes the parent of the \texttt{\currbb}, if it exists. The \texttt{⟨node⟩} is advanced by one using the \texttt{\addbookmark} macro.

The \texttt{\bookmarks} macro opens the group, redefines the \texttt{\doctype} and \texttt{\ptocentry} (i.e. macros from \texttt{\tocbuffer}), inserts the first PDF outline with the name of the document and executes the \texttt{\tocbuffer}.

The \texttt{\setoutline \{⟨label⟩\}\{⟨text⟩\}\{⟨prefix⟩\}} creates the PDF outline \texttt{⟨prefix⟩⟨text⟩} and the link with \texttt{⟨label⟩} is activated. The \texttt{\tempnum} register includes the number of children of this PDF outline.

The special “conversion” macro \texttt{\cnvbookmark} is used here. It is nonactive by default. User can set (for example) \texttt{\let\cnvbookmark=\lowercase} for \texttt{č} → \texttt{c}, \texttt{ž} → \texttt{z} etc. conversions. The \texttt{\icc} setting can be done by \texttt{\markskhook}.

The text is converted by \texttt{\nobraces} macro for removing \texttt{\{}. The macro \texttt{\nobrA} is used here too. When we have (for example) \texttt{\{\tt\text{⟨text⟩}\in\TeX\}} then the \texttt{\text{⟨text⟩}\in\TeX} is the result of such conversion.

\section{5.11 Sorting by Alphabetical Order}

This work is done by \texttt{\sortindex} macro. First version implemented the bubble sort algorithm but it was slow for large indexes. For example sorting of the index of this document has taken circa

\texttt{\ignoretwo: 31, 37, 39, 42 \remakebackslash: 39 \addbookmark: 38–40 \currbb: 38–40 \currssec: 38 \bookmarks: 7, 36, 40 \setoutline: 40 \cnvbookmark: 14, 40 \nobraces: 40 \nobrA: 40}

40
2 seconds of computer time. My son Mirek rewrote the sorting by mergesort algorithm in the second version of docByTeX. The previous 52 thousand sorting queries (for an index of the size comparable with the index used here) was reduced to 1600 queries, so 30 times better.

First, we declare the \isAleB which answers true if A < B (see also \isAleB macro below). The auxiliary macros \nullbuf, \return and \fif are used here. The \return macro is used for escaping from various loops to the \relax mark. The \fif are balanced by the \fif macro in nested \fif constructions. This save the number of \expandafter commands.

The \sortindex macro puts to the input queue the content of the whole \indexbuffer followed by \end, \end, the new \indexbuffer is set as empty and the \mergesort macro is executed.

The \mergesort takes two groups of items repeatedly, each group is sorted already. The groups are separated by commas in the input queue. These two groups are merged to one sorted group. This process is repeated until \end occurs. One merging of two groups looks like that: suppose for example two groups eimn,bdkz, which is merged to one group bdeikkmmz,. Letters in that example are the whole sorted entries.

At the begin of the process, all groups have only one item. After first pass over input queue, the result is the groups with two items. They are saved back in the \indexbuffer. Next pass puts the \indexbuffer to the input queue and creates groups with four items. Next, there are 8 items per group etc. This process is repeated until only one sorted group is created (line 935) and only \end is in the second group. The \gobblerest macro removes the second \end from input queue.

The core of the \mergesort is on the lines 942–947. The \mergesort macro saves first item of the first group to the #1 parameter, next items of the first group to the #2 parameter and the first item of the second group to the #3 parameter. If #1 < #3 then we save #1 to the output \indexbuffer, the #1 is removed from input queue and \mergesort is executed again. The cases with empty parameters are solved in the lines 926–932: we need to save the rest of the nonempty group to the output \indexbuffer.

\nullbuf: 41 \return: 41 \fif: 41 \sortindex: 39–41 \mergesort: 41–42

//isAleB: 41 \nullbuf: 41 \return: 41 \fif: 41
and go to the next pair of groups. If the terminal string `\end,\end` is scanned then the next run of \mathit{\texttt{mergesort}} is executed after \texttt{indexbuffer} is put to the input queue and set it to empty value.

The sorting of the two items are realized by \texttt{\isAleB (itemA)(itemB)} macro. The items are in the form `\langle \text{wordA} \rangle \langle \text{wordB} \rangle`. The macro converts these parameters to the strings by \texttt{\string} primitive and expands to `\testAleB \langle \text{wordA} \rangle \text{relax} \langle \text{wordB} \rangle \text{relax}`. The \texttt{\lowercase} primitive is executed here because we needn’t distinguish between uppercase/lowercase letters.

\begin{verbatim}
\def\isAleB #1#2{
  \edef\tmp{\expandafter\ignoretwo\string#1&0\relax\expandafter\ignoretwo\string#2&1\relax}
  \lowercase\expandafter{\expandafter\testAleB\tmp}
}
\end{verbatim}

The \texttt{\testAleB \langle \text{wordA} \rangle \text{relax} \langle \text{wordB} \rangle \text{relax}} macro tests if `\langle \text{wordA} \rangle` precedes `\langle \text{wordB} \rangle`. If the first letters are the same, the macro is called recursively. The recursion will be truly finished because different tails are appended to the compared words at line 952.

\begin{verbatim}
\def\testAleB #1#2\relax #3#4\relax {
  \ifx#1#3\testAleB #2\relax #4\relax \else \AleBtrue \else \AleBfalse \fi
}
\end{verbatim}

The macro \texttt{\napercarky} inserts \, separators between items in the sorted \texttt{indexbuffer}.

\begin{verbatim}
\def\napercarky{\ifx#1\end \else \noexpand,\noexpand#1\noexpand\expandafter\napercarky \fi}
\end{verbatim}

5.12 Merging of the List of the Page Numbers

Each occurrence of the `\langle \text{word} \rangle` is stored to the \texttt{\reffile} as \texttt{\refuseword \{\langle \text{word} \rangle\}\{\langle \text{page} \rangle\}}. This macro is processed at the begin of the document when \texttt{\reffile} is read.

\begin{verbatim}
\def\refuseword#1#2{
  \expandafter\expandafter\expandafter\transf\csname w:#1\endcsname,0,\csname w:#2\endcsname
}
\end{verbatim}

So, the list of the pages where the `\langle \text{word} \rangle` occurs is stored in the \texttt{\w:\langle \text{word} \rangle} macro. Pages are separated by commas. The list looks like:

2,5,10,11,12,13,13,27

We need to convert this list to the format 2,5,10--13,27, i.e. we need to remove double occurrences and to replace consecutive lists of pages by intervals in the form `\langle from\rangle--\langle to\rangle`. This work is done by \texttt{\listofpages \langle \text{word} \rangle} macro which puts the list of pages to the input queue terminated by `,0`, and executes the \texttt{\transf} macro.

\begin{verbatim}
\def\listofpages#1{
  \expandafter\expandafter\expandafter\transf\csname w:#1\endcsname,0,\csname w:#1\endcsname
}
\end{verbatim}

The \texttt{\transf} macro removes the page numbers which are equal to \texttt{\dgnum} or \texttt{\apinum}. We want to avoid the double occurrence of the main page and underlined page in the list. These pages are printed separately. The declaration of the registers follows:

\begin{verbatim}
\def\dgnum#1#2{
  \def\apinum#1#2{
  \def\tempnum#1#2{
  \def\dash#1{
    \def\isAleB:41–42 \texttt{\isAleB:41–42} \texttt{\napercarky:41–42} \texttt{\refuseword:25,30,37–38,42} \texttt{\listofpages:20–21,42} \texttt{\dgnum:20–21,42–43} \texttt{\apinum:20–21,42–43}}
\end{verbatim}
The \tempnum is current page number processed in the list and \ifdash returns true if the interval is opened by ⟨from⟩. The \iffirst returns true if the first page of the list is processed.

The \transf \{list of pages\},0, executes repeatedly the \cykltransf macro.

\transf 42–43 \cykltransf: 43

Multicolumn typesetting

The macros for multicolumn printing are borrowed from “\TeXbook inside out”, pages 244–246.
One problem is solved in addition. We check the empty space on the current page before the section title is printed. This work is done by \calculatedimone (executed by \doindex macro at the line 862).

The final settings, catcodes

The catcodes are set at the end of the docby.tex file. We add the active category for the " character and we set the _ as a normal character because this character is mostly used in the identifiers and the catcode 8 of this character causes many problems.

The \everymath and \everydisplay returns the category of _ to the plain\TeX meaning (as math index prefix).

The active " character separates the “inline verbatim” environment.

The \langleactive sets the active catcode for the < char. So, you can write <text> in “inline verbatim” and the \text is printed.

Index

The control sequences marked by (>) are sequences at user level. Other control sequences are internal in DocBy.T\TeX. The bold page number points to the place where the sequence is defined and documented, other page numbers point to occurrence of the sequence. The control sequences for users have underlined pagernumber in the list of page numbers. This means the page where the sequence is documented at user level.
\ignorerelax: 38
\ignoretwo: 40, 31, 37, 39, 42
\iid: 31, 10, 20, 32
\iidgh: 32, 10
\iidgm: 32, 10
\iidl: 31, 10, 32
\iidlh: 32, 10
\iidln: 32, 10
\ilink: 35
\ilpart: 35
\isec: 34, 35
\issubsec: 35, 34
\stitle: 18, 19
\ilabel: 28, 9, 27
\ilabellee: 28
\ilabellist: 28, 27
\inchquote: 23, 8
\indexbuffer: 39, 33, 37-38, 41-42
\indexbook: 14, 39
\inxnext: 26, 7, 8-9, 14-15, 21-22
\inputfilename: 25, 21-22, 26-27
\ins: 15, 1, 7, 13
\insinternal: 26, 25, 27
\isAliB: 42, 41
\isnameprinted: 22
\istocsec: 38
\item: 23, 13
\itemno: 23, 13
\itsmall: 16, 17
\label: 36, 12, 13, 31-32, 37
\labeltext: 36, 13, 28, 35
\langleactive: 44, 14
\lastline: 28, 26-27
\leftcomment: 15, 26, 44
\linecomment: 15, 26, 44
\lineno: 25, 8, 27-28
\linkskip: 35, 18, 36
\listofpages: 42, 20-21
\locword: 29, 39
\lword: 30, 26, 29
\makelinks: 34, 17-18, 35
\managebrackets: 31, 30
\maybe: 31
\mergesort: 41, 42
\module: 15, 4, 10-14
\modulename: 15
\mydotfill: 21
\mydotdots: 21
\namespace: 29, 11, 15
\namespacemacro: 29, 30
\napercarky: 42, 41
\nb: 23, 8, 11, 15, 30, 34, 36, 39-40
\nextparam: 31, 30
\noactive: 24, 6, 15, 29, 32
\nobra: 40
\nobraces: 40
\backmatter

\section*{Index}

\begin{itemize}
\item \texttt{\tochook}: 14, 39
\item \texttt{\totalfoocount}: 33, 37
\item \texttt{\totalfoodim}: 33, 37
\item \texttt{\transf}: 43, 42
\item \texttt{\ttlineno}: 25, 29
\item \texttt{\ttsmall}: 16, 17, 20–22, 33
\item \texttt{\ttstrut}: 16, 17, 21–22
\item \texttt{\undefined}: 23, 20–21, 25, 28–30, 32, 36, 40
\item \texttt{\varparam}: 31, 30
\item \texttt{\vsize}: 16, 33, 43–44
\item \texttt{\writelabel}: 36
\item \texttt{\writelabelinternal}: 36
\item \texttt{\Yellow}: 17, 19–20, 22
\end{itemize}

\begin{verbatim}
struct mypair
\end{verbatim}