version May 2014

DocBy.TEX – Making a Documentation Of Sources By TEX

Petr Olšák

www.olsak.net/docbytex.html

Table of Contents

1 Preface

DocBy.TEX gives you a possibility to creating a documentation of source codes by TEX. The source codes can be 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 it in second. Now, there is no need to merge both information (for computer and for human being) to single file.

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 needs to document another computer language, you can change some macros (see the section 3).

Wee 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 `//:␣`⟨*keyword*⟩ line. The example follows.

Suppose that the following text is written in the file `foo.d`

```
The struct \dg [struct] mypair is used as a return value of
"my_special_function".  There are two "float" values.
\ins c mypair

The \dg [struct mypair] my_special_function() has one parameter "p"
and returns double and triple of this parameter in "mypair" struct.
\ins c my_special_function
```

The file `foo.c` has to include the comments `//:␣mypair` and `//:␣my_special_function`. These comments delimit the part of source code to be listed in the documentation:

```
#include <stdio.h>

//:  mypair

struct mypair {
  float x, y;
};

//:  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;
}
```

The result looks like that:

The struct mypair is used as a return value of my_special_function. There are two float values.

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

The my_special_function has one parameter p and returns double and triple of this parameter in mypair struct.

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: }
```

The first listed part of source code is started by `//:␣mypair` and ended by firs occurrence of the `//:`. The second listed part is started by `//:␣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 `#include␣<stdio.h>` has number one and first printed line has the number five.

The `//:␣`⟨*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.

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 pdfTEX extended by encTEX. 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 encTEX is detected. Otherwise (if encTEX 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 encTEX 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 pdfTEX 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 encTEX 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 EncTEX

The hyperlinked words are located by encTEX by "hungry algorithm". It means that if there are two documented words abc and abcde then the text abcdefg is divided to the hyperlinked part abcde (the blue color is used) and to the normal part fg (black color). The hyperlinked part points to the place of the documentation of the word abcde. On the other hand the text abcdx is divided to hyperlinked part abc and this part points to the documentation of the word abc.

EncTEX 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. EncTEX 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 ⟨*word*⟩ as a searched word (for encTEX) 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 oputput 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:

```
#!/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:

```
why:    --   don't print first nor ending line
why:    +-   print first line but don't print ending line
why:    -+   don't print first line but print ending line
```

```
why:    ++   print both lines
```

If the parameter ⟨*from*⟩ is empty (use `{}` notation) then the printing starts on 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. The ending line does not exists in such case.

If the parameter ⟨*from*⟩ (or ⟨*to*⟩ respectively) has `\empty` value (use `{\empty}` notation) then the printing starts (or stops respectively) at the first empty line. You can specify if this line is printed by ⟨*why*⟩ parameter.

The parameters ⟨*from*⟩ and ⟨*to*⟩ can be started by `^^B` character (it means that the pattern have to be at the begin of the line) and/or they can be ended by `^^E` character (it means that the pattern have to be at the end of line). For example the parameter `^^Btext^^E` means that `text` have to be on the line without nothing more.

The special TEX characters (special categories) are not allowed in ⟨*from*⟩ and ⟨*to*⟩ parameters. You have to use special control sequences `\nb`, `\obrace`, `\cbrace`, `\percent` and `\inchquote` instead of `\`, `{`, `}`, `%`, `"` characters. You can define aditional sequences for another special TEX characters, for example:

```
{\catcode`\#=12 \gdef\hashmark{#}}
```

If parameters ⟨*from*⟩ and ⟨*to*⟩ are the same or the ⟨*from*⟩ pattern is on the same line as ⟨*to*⟩ pattern then only this line is printed (⟨*why*⟩ have to be `++` or `+-`). If this condition is true but ⟨*why*⟩ is `-+` or `--`, then the printing of the code is stopped at next line with ⟨*to*⟩ pattern or at the end of the file.

The `\ifirst` 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{`⟨*from*⟩`}{`⟨*to*⟩`}{`⟨*why*⟩`}`. This command starts the searching of the ⟨*from*⟩ pattern from the first line which wasn't read by the previous `\ifirst` or `\inext` command. The parameters of the `\inext` command have the same meaning as the parameters of the `\ifirst` command. The parameter ⟨*file*⟩ is missing because the ⟨*file*⟩ from the last `\ifirst` command is used.

The number of the last line read by `\ifirst` 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:

```
\ifirst {file.txt}{foo}{foo}{++}        % print the first line
                                        % with the text "foo"
\inext {foo}{foo}{++}                   % print the next line with
                                        % the occurence of "foo"
\ifirst {file.c}{//:  from}{//:}{--}     % the same as \ins command
\ifirst {file.h}{func(}{)}{++}          % print of function prototype
\ifirst {file.c}{func(}{^^B\cbrace}{++} % print of the code func
\ifirst {file.txt}{}{\end}{++}          % print of the whole file
\ifirst {file.txt}{}{\empty}{+-}        % print of the first block
                                        % 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*⟩␣`-`␣`1` means how many occurrences 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 \skippingfalse. If the ⟨*why*⟩ parameter have different value and/or \skipingtrue then you must add/subtract one or two to/from the line number/number of lines. Examples:

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

2.7 References to Line Numbers

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

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

The ⟨*label*⟩ have to be unambiguous in the whole document. The \cite reference works forward and backward (after second pass of TEX run).

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

DocBy.TEX does not write any warning if a ⟨*text*⟩ doesn't occur in the listing. Of course, if you use the unconnected ⟨*label*⟩ by \cite command then the warning is printed.

The following example uses the known file foo.c mentioned in the section 2.2.

```
The declaration of my very special function is on the line~\cite[myfunct].

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

There is very specific idea on the line~\cite[returnx] where the input
parameter is multiplied by two.
```

2.8 Verbatim Environment by \begtt/\endtt and by Quotes

Verbatim displays of the code can be included to the documentation by \begtt and \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 $...$.

2.9 The Declaration of the Documented Word

You can use commands \dg, \dgn, \dgh, \dl, \dln or \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 \dg command.

The possibilities of the syntax follows:

```
\dg ⟨word⟩              % ⟨word⟩ separed by space
\dg [⟨text⟩] ⟨word⟩     % optional paremeter ⟨text⟩
\dg [⟨text⟩]⟨word⟩      % the space between [⟨text⟩] add ⟨word⟩ is optional
\dg ⟨word⟩()            % ⟨word⟩ with "()" separed by space
\dg [⟨text⟩]⟨word⟩()    % a combination of previous syntax
\dg ⟨word⟩,             % ⟨word⟩ separed by comma
\dg [⟨text⟩] ⟨word⟩,    % a combination of previous syntax
\dg ⟨word⟩(),           % ⟨word⟩ with "()" separed by comma
\dg [⟨text⟩]⟨word⟩(),   % a combination of previous syntax
\dg ⟨word⟩.                      % ⟨word⟩ separed by period
etc...
```

In general: The optional `[` can follow after `\dg` command. The ⟨*text*⟩ separated by `]` is read in such case and subsequent optional space is moved to the end of the ⟨*text*⟩. It means that `\dg [text]␣word` is the same as `\dg [text␣]word`. Next, the ⟨*word*⟩ is read. The ⟨*word*⟩ parameter cannot include the space, comma, period, colon and semicolon because these characters can be separator of the ⟨*word*⟩. These punctuation characters are not part of the ⟨*word*⟩ but they are printed. It means that `\dg␣word:` prints `word:` to the output and sets the `word` as a documented word. If the scanned ⟨*word*⟩ ends by brackets `()` then these brackets are removed from ⟨*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 ‘ (backward quote) then the space and this quote is not printed.

Examples: `\dg` ⟨*word*⟩␣‘~⟨*next-text-without-line-breaking*⟩ or: `\dg` ⟨*word*⟩␣‘`"..."`.

The commands `\dgh`, `\dgn`, `\dln`, `\dlh` with space as a separator doesn't print this separator because they usually print nothing (see below).

Semantic: The ⟨*word*⟩ parameter is documented word. If this ⟨*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 `\dg` command is used and the optional ⟨*text*⟩ or `()` does not printed. This is the destination of all blue hyperlinks. The ⟨*word*⟩ is printed in footnote of the current page too including the optional ⟨*text*⟩ in and/or including the optional `()`. 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 ⟨*word*⟩, not by the optional ⟨*text*⟩.

The ⟨*word*⟩ declared by `\dg` is declared globally. This place is a reference point for the whole document.

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

The `\dgn` command (mnemonic: `\dg` next) saves its parameters but prints nothing. The first occurrence of the ⟨*word*⟩ in the next listing will be treated as the `\dg` is written here.

The `\dl` declares ⟨*word*⟩ locally. If the short name ⟨*word*⟩ is used in the same name space then it is hyperlinked and pointed to the place where `\dl` is used. The name space is changed by `\module` command. It means that ⟨*word*⟩ is used locally in the module. The word declared by `\dl` lives in two variants: short name “⟨*word*⟩” and long name (depends on the current name space, typically “⟨*word*⟩`./`⟨*modulename*⟩”). 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 most once in the document else the error is printed by DocBy.TEX on the terminal. In case of `\dl` the short name is irrelevant but the long name have to be unambiguous.

The `\dlh` command is `\dl` hidden and the `\dln` means `\dl` next. They are similar as `\dgh` and `\dgn`.

If somebody hate this complicated parameter scanning then he/she can use internal commands with three parameters in braces: `\iidg`, `\iidgh`, `\iidgn`, `\iidl`, `\iidlh`, `\iidln`. The usage of the parameters is: `\iidg{`⟨*text*⟩`}{`⟨*word*⟩`}{`⟨*brackets*⟩`}`. Of course, you can do more by these commands: you can declare ⟨*word*⟩ with spaces or another delimiters, you can write something different than “`()`” as ⟨*brackets*⟩ parameter.

2.10 Namespaces

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

The namespace is undefined out of \namespace...\endnamespace environment thus the \dl command cannot be used here. The \module command declares namespace #1./⟨*modulename*⟩ thus you can use \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:

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

The \namespace...\endnamespace environments can be nested. The inner environment have to have another namespace than the outside environment. These environments work globally independent of the \bgroup and \egroup. The \endnamespace command used outside of all namespace environments does nothing. You needn't to close these environments before \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 ⟨*word*⟩ (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 \dg command (or similar). The second place where the word is documented only for users without code can be declared by \api{⟨*word*⟩}. 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 \api command is used. Of course, the \api{⟨*word*⟩} command is not sufficient to including the word to the index. You need use the \dg command (in another place of the document) too.

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

You can reference the place marked by \api{⟨*word*⟩} by \cite[+⟨*word*⟩]. This macro expands to the page number where the \api{⟨*word*⟩} is used. For example the \cite[+\nb␣api] expands to 11 in this document.

If there exist the API destination declared by \api command then the red word printed in the \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 ⟨*Section-Name*⟩\par command. Each section can include subsections started by the command \subsec ⟨*Subsection-Name*⟩\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 ⟨*Part-Name*⟩\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␣⟨*modulename*⟩␣ command creates a new section Module␣⟨*modulename*⟩, creates namespace and includes the ⟨*modulename*⟩.d file. You can change this default behavior, see sections 3.1 and 3.3.

The \title⟨*Name*⟩\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⟨*text*⟩\par command centers the ⟨*text*⟩ 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 ⟨*label*⟩ in square brackets can be used with \sec and \subsec commands. The parameters looks like: \sec [⟨*label*⟩]␣⟨*Section-Name*⟩\par. If the ⟨*label*⟩ parameter is used then you can reference this place by \cite[⟨*label*⟩]. This macro prints the number of referenced (sub)section and acts like hyperlink.

You can disable the transport of ⟨*(Sub)Section-Name*⟩ into table of contents by \savetocfalse used before \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 ⟨*label*⟩. This can be done by optional parameter of the \sec or \subsec command (see the section 2.12) or by the command \label[⟨*label*⟩] 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[⟨*label*⟩] expands to the number of the page where the ⟨*label*⟩ is. The command \numref[⟨*label*⟩] expands to the result which depends on the type of the destination:

- sections number if the destination is the section
- the pair ⟨*secnumber*⟩.⟨*subsecnumber*⟩ 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 [⟨*label*⟩]{⟨*text*⟩} in order to create the hyperlink in PDF mode. This macro prints the ⟨*text*⟩ in blue color and it is treated as hyperlink to the destination specified by ⟨*label*⟩. For example the command \cite[⟨*label*⟩] does the same as \ilink[⟨*label*⟩]{\numref[⟨*label*⟩]}. The real macro \cite executes a test if the \numref[⟨*label*⟩] is empty and prints the \pgref in such case.

If the ⟨*label*⟩ is not declared then \pgref{⟨*label*⟩} and \numref{⟨*label*⟩} 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 ⟨*modulename*⟩␣ command creates the section with the label "m:⟨*modulename*⟩". You can reference it by:

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

The \refmodule{⟨*modulename*⟩} defined in the example above prints ⟨*modulename*⟩ and creates it as hyperlink. For example \refmodule[base] prints the word "base" in blue typewriter font and creates it as the hyperlink to the begin of the section "Module base" if this section is crated by \module␣base␣ command.

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

The \api{⟨*word*⟩} command executes \label[+⟨*word*⟩] internally. It means that you can reference this place by \ilink[+⟨*word*⟩]{API:␣⟨*word*⟩} for instance.

No more automatic numbering is processed by DocBy.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[⟨*label*⟩]{⟨*text*⟩} command in such case. This macro expands it parameters immediately and inserts invisible hyperlink destination into typeset material in horizontal mode. Then macro \numref{⟨*label*⟩} expands to ⟨*text*⟩ in the next pass of the TEX run. Example: we define the macro \bib[label] which inserts the destination marked by the ⟨*label*⟩. The hyperlink with the number of the book can be created by \cite[b:⟨*label*⟩].

```
\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 \ifig ⟨*width*⟩␣⟨*picname*⟩␣ inserts the picture into your document. The picture have to be prepared in the file fig/⟨*picname*⟩.eps (if DVI mode is used) and in the file fig/⟨*picname*⟩.pdf (if PDF mode is used). You can use another directory for pictures than fig/ – this name is stored in the \figdir macro and you can redefine it. The ⟨*width*⟩ parameter is the ratio of the width of inserted picture to the \hsize (unit-less). The inserted picture is placed to left side with the paragraph indentation. For example \ifig␣0.5␣foo inserts the picture prom 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 eps format and you need to convert it to pdf then you can use:

```
ps2pdf -dEPSCrop ⟨picname⟩.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 ⟨*mark*⟩␣. The ⟨*mark*⟩ is printed left from the text. If the ⟨*mark*⟩ is the star (*) then it is changed to the bullet. You can write \item \the\itemno)␣ if you want to print numbered items. The \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 plainTEX macro \item then just don't use \begitems and \enditems commands.

3 For Advanced Users

The definitions of basis macros of DocBy.TEX are mentioned in this section. The user can change these definitions if he need different behavior of DocBy.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.

3.1 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 \projectverion 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.

docby.tex

```
\def\titmodule{Module}
\def\tittoc{Table of Contents}
\def\titindex{Index}
\def\titversion{version }
\def\opartname{>> PART}
\ifx\chyph\undefined
\else \ifnum\language=0
      \else
          \def\titmodule{Modul}
          \def\tittoc{Obsah}
          \def\titindex{Rejstřík}
          \def\titversion{verze }
          \def\opartname{>> CAST}
\fi   \fi
```

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.

docby.tex

```
\def\begtthook{}
\def\quotehook{}
\def\indexhook{}
\def\tochook{}
\def\bookmarkshook{}
\def\outputhook{}
```

The \begtthook macro is inserted after begin of the group and after all catcodes are set by default before text inside the environment \begtt...\endtt is processed. The \quotehook macro is inserted after begin of the group and after all cactodes 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{\langleactive} % <text> is changed to ⟨text⟩
\def\begtthook{\mubytein=1}   % auto-hyperlinks between \begtt...\endtt
\def\begtthook{\setsmallprinting} % \begtt...\endtt printed by small font
\def\begtthook{\catcode`\!=0} % !commands can be used in \begtt...\endtt
\def\indexhook{The special index with such and such properties follows.}
```

\titindex: 14, 39 \tittoc: 14, 39 \titmodule: 14–15 \titversion: 14, 19 \opartname: 14, 40
\begtthook: 14, 28 \quotehook: 14, 44 \indexhook: 14, 39 \tochook: 14, 39
\bookmarkshook: 14, 40 \outputhook: 14–15, 34

```
\def\outputhook{\let\mylogo=\relax} % \mylogo is not expaded in *.ref
```

3.3 The Commands \module and \ins

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

docby.tex

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

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

docby.tex

```
55: \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 //..⟨*eol*⟩ and /*␣..␣*/. These comments are printed in green color. You can disable this behavior by \noactive⟨*string*⟩ command. You can set a new type of comments by \setlinecomment {⟨*string*⟩} command. These commands will be turned to green color from ⟨*sting*⟩ to end of line. These commands work globally. For example

```
\noactive{/*}\noactive{*/}\noactive{//}
\setlinecomment{\percent} \noactive{\nb\percent}}
```

activates comments used in TEX sources and PostScript language.

You can set the comments of the type /*...*/ by the command \setlrcomment {⟨*left*⟩}{⟨*right*⟩}.

If you are interested what these macros do internally then you can read the following part of this section.

docby.tex

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

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

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

docby.tex

```
68: \def\linecomment {\let\Black=\Green \Green}
69: \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 encTEX 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

```
71: \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 51 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:

docby.tex

```
73: \setlinecomment{//}  \setlrcomment{/*}{*/}
```

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.

docby.tex

```
77: \parindent=30pt
```

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

docby.tex

```
79: \newdimen\nwidth \nwidth=\hsize \advance\nwidth by-2\parindent
```

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).

docby.tex

```
81: \raggedbottom
82: \exhyphenpenalty=10000
```

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

docby.tex

```
84: \font\bbf=csb10 at12pt
85: \font\bbbf=csb10 at14.4pt
86: \font\btt=cstt12
87: \font\ttsmall=cstt8
88: \font\rmsmall=csr8
89: \font\itsmall=csti8
90: \font\partfont=csb10 at80pt
```

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-artifacts occur at background of listings. The `\setnormalprinting` is similar.

\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

docby.tex
```
\def\setsmallprinting{\ttsmall \let\it=\itsmall \let\rm=\rmsmall
   \def\ttstrut{\vrule height8pt depth3pt width0pt}%
   \offinterlineskip \parskip=-1pt\relax
}
\def\setnormalprinting{\tt \baselineskip=0pt \hfuzz=4em
   \def\ttstrut{\vrule height10pt depth3pt width0pt}%
   \offinterlineskip \parskip=-1pt\relax
}
```

The design is projected only with the following colors: \Blue, \Red, \Brown, \Green, \Yellow a \Black. If you need other colors you can define more similar macros.

docby.tex
```
\ifx\pdfoutput\undefined
   \def\setcmykcolor#1{}
\else
   \def\setcmykcolor#1{\special{PDF:#1 k}}
\fi
\def\Blue{\setcmykcolor{0.9 0.9 0.1 0}}
\def\Red{\setcmykcolor{0.1 0.9 0.9 0}}
\def\Brown{\setcmykcolor{0 0.85 0.87 0.5}}
\def\Green{\setcmykcolor{0.9 0.1 0.9 0.2}}
\def\Yellow{\setcmykcolor{0.0 0.0 0.3 0.03}}
\def\Black{\setcmykcolor{0 0 0 1}}
\let\oriBlack=\Black
```

All colors are defined by \setcmykcolor macro which is empty in DVI mode but a proper \special is used in PDFTEX 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.

docby.tex
```
\ifx\pdfoutput\undefined
   \def\rectangle#1#2#3#4{\vbox to0pt{\vss\hrule\kern-.3pt
      \hbox to#3{\vrule height#1 depth#2\hss#4\hss\vrule}%
      \kern-.3pt\hrule\kern-#2\kern-.1pt}}
\else
   \def\rectangle#1#2#3#4{\vbox to0pt{\vss\hbox to#3{%
       \rlap{\Yellow \vrule height#1 depth#2 width#3}%
       \hss#4\Black\hss}\kern-#2}}
\fi
```

The DocBy.TEX logo is typeset by \docbytex macro.

docby.tex
```
\def\docbytex {\leavevmode\hbox{DocBy}.\TeX}
```

4.2 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 \docbytex: 17, 34, 40 \printsec: 17–18, 34–35
\printsecbelow: 18, 34–35

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 \noberak. 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 ⟨*label*⟩ 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.

docby.tex

```
\def\printsec #1{\par
    \removelastskip\bigskip\medskip
    \noindent \makelinks
    \rectangle{16pt}{9pt}{25pt}{\Brown\bbbf\ifsavetoc\the\secnum\else\emptynumber\fi}%
    \kern5pt{\bbbf\let\_=\subori #1}\par
}
\def\printsecbelow {\nobreak\medskip}
```

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

docby.tex

```
\def\printsubsec #1{\par
    \removelastskip\bigskip
    \noindent \makelinks
    \vbox to0pt{\vss
       \rectangle{16pt}{9pt}{25pt}{\Brown\bf
          \ifsavetoc\the\secnum.\the\subsecnum\else\emptynumber\fi}\kern-5pt}%
    \kern5pt{\bbf\let\_=\subori \let\tt=\btt #1}\par
}
\def\printsubsecbelow {\nobreak\smallskip}
```

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.

docby.tex

```
\def\printpart #1{\par
    \removelastskip\bigskip\medskip
    \noindent {\linkskip=60pt\makelinks}%
    \rectangle{16pt}{9pt}{25pt}{}%
    \kern-20pt{\Brown\partfont\thepart\Black}\kern10pt{\bbbf #1}\par
}
\def\printpartbelow {\nobreak\bigskip}
```

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

docby.tex

```
\def\emptynumber{}
```

4.3 The Title, The Author

The \title ⟨*title*⟩\par macro reads its parameter ⟨*title*⟩ 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 ⟨*title*⟩ is stored into \headtitle macro (in both modes) only if the \headtitle is empty, it means that it it not initialized by user. Then \iititle suppresses the headline printing on the current page by the \nohedaline command.

docby.tex

```
\def\title{\def\tmpA{title}\futurelet\nextchar\secparam}
\ifx\pdfoutput\undefined
   \def\iititle {\par
      \ifx\headtitle\empty\edef\headtitle{\the\sectitle}\fi
      \noheadline
```

\printsubsec: 18, 35 \printsubsecbelow: 18, 35 \printpart: 18, 35 \printpartbelow: 18, 35
\emptynumber: 12, 18, 35 \title: 12, 4, 18–19 \iititle: 18–19

```
163:       \ifx\projectversion\empty \else
164:          \line{\hss\rmsmall\titversion\projectversion}\nobreak\medskip\fi
165:       \centerline{\bbbf \let\_=\subori\the\sectitle}\nobreak\medskip}
166: \else
167:   \def\iititle {\par
168:       \ifx\headtitle\empty\edef\headtitle{\the\sectitle}\fi
169:       \noheadline
170:       \indent\rlap{\rectangle{25pt}{15pt}{\nwidth}{\Black\let\_=\subori\bbbf\the\sectitle}}%
171:           \ifx\projectversion\empty \else
172:           \hbox to\nwidth{\hss
173:              \raise26pt\hbox{\Brown\rmsmall\titversion\projectversion\Black}}\fi
174:       \par\nobreak\vskip15pt}
175: \fi
```

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.

docby.tex

```
177: \ifx\projectversion\undefined \def\projectversion{}\fi
```

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

docby.tex

```
179: \def\author #1\par{\centerline{\bf #1\unskip}\smallskip}
```

4.4 Headers and Footers

DocBy.TEX doesn't change the output routine defined by plainTEX. It uses the standard plainTEX'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.

docby.tex

```
183: \footline={\hss\rectangle{8pt}{2pt}{25pt}{\tenrm\Black\folio}\hss}
```

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.

docby.tex

```
185: \headline={\normalhead}
186: \def\normalhead {\savepglink \let\_=\subori
187:     \vbox to0pt{\vss \baselineskip=7pt \lineskiplimit=0pt
188:      \line{\indent\Black\tenit \firstmark \hss \headtitle\indent}
189:      \line{\indent\Yellow\xleaders\headlinebox\hfil\indent\Black}}}
```

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.

docby.tex

```
191: \def\noheadline {\global\headline={\savepglink\hfil\global\headline={\normalhead}}}
```

The \headtile 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.

docby.tex

```
193: \ifx\headtitle\undefined \def\headtitle {}\fi
```

The auxiliary macro \headlinebox prints the empty rectangle in DVI mode and solid yellow rectangle in PDF mode. It is used on the line 189 for creating of square filled line in the header.

\projectversion: 12, 19 \author: 12, 4, 19 \footline: 19 \headline: 19, 36 \normalhead: 19
\noheadline: 18–19 \headtile: 40 \headlinebox: 19–20

docby.tex

```
\ifx\pdfoutput\undefined
   \def\headlinebox{\hbox{\kern2pt\rectangle{4pt}{0pt}{4pt}{}\kern2pt}}
\else
   \def\headlinebox{\hbox{\kern2pt\vrule height4pt depth0pt width4pt\kern2pt}}
\fi
```

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.

docby.tex

```
\ifx\pdfoutput\undefined
   \def\printdg#1#2#3{\leavevmode\hbox{\kern-.6pt
      \vbox{\hrule\hbox{\vrule height8.5pt depth2.5pt \kern.2pt
            \tt#2\kern.2pt\vrule}\hrule\kern-2.9pt}\kern-.6pt}}
\else
   \def\printdg#1#2#3{\leavevmode \setbox0=\hbox{\tt#2}%
           \Yellow\rlap{\vrule height8.7pt depth2.7pt width\wd0}%
           \printdginside{#2}{\box0}}
\fi
```

The red text is printed by auxiliary macro \printdginside. 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.

docby.tex

```
\def\printdginside#1#2{\ifnum\pgref[+#1]>-1 {\let\Blue=\Red \ilink[+#1]{#2}}%
    \else \Red#2\relax\Black\fi}
```

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.

docby.tex

```
\def\printfnote #1#2#3#4{%
  \specfootnote{{\let\Black=\oriBlack \ttsmall #1\Red #4\Black#3\rmsmall
       \apinum=\pgref[+#2]\relax
       \ifnum\apinum>-1 :~\lower1.4pt\vbox{\hbox{\pglink\apinum}\kern1pt\hrule}\fi
       \undef{w:#2}\iftrue \setbox0=\hbox{}\else \dgnum=-1 \setbox0=\hbox{\listofpages{#2}}\fi
       \ifdim\wd0=0pt \else
           \ifnum\apinum>-1 , \else :~\fi
           \unhbox0
       \fi}}%
}
```

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.TEX's default design, only one \indent more in \ptocsubline:

\printdg: 20, 31–32 \printdginside: 20, 32 \printfnote: 20, 31–32 \ptocline: 21, 38
\ptocsubline: 20–21, 38

docby.tex

```
\def\ptocline #1#2#3{%
   \if^^X#1^^X\advance\partnum by1 \medskip \fi
   \line{\rectangle{8pt}{1pt}{25pt}{%
     \if^^X#1^^X\ilink[sec:\thepart]{\bbbf \thepart}\else\ilink[sec:#1]{#1}\fi}\kern5pt
     {\bf\let\_=\subori #2}\mydotfill\pglink#3}}
\def\ptocsubline #1#2#3{%
   \line{\indent\rectangle{8pt}{1pt}{25pt}{\ilink[sec:#1]{#1}}\kern5pt
         \let\_=\subori #2\mydotfill\pglink#3}}
\def\mydotfill{\leaders\hbox to5pt{\hss.\hss}\hfil}
```

The \mydotfill command prints the dots in table of contents so they are aligned.

The \ptocentry ⟨*type*⟩{⟨*word*⟩}{⟨*s-word*⟩} prints one item about documented word in table of contents. If it is \api occurrence of the ⟨*word*⟩ then ⟨*type*⟩=+ else ⟨*type*⟩=@. The ⟨*s-word*⟩ parameter is empty but if the ⟨*word*⟩ is declared by \dl 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.

docby.tex

```
\def\ptocentry#1#2#3{\ifhmode,\hskip 7pt plus 20pt minus 3pt \fi
    \noindent \hbox{\ttsmall \if+#1\apitext\fi \ilink[#1#2]{\ifx^^X#3^^X#2\else#3\fi}}%
    \nobreak\myldots\pglink\pgref[#1#2]\relax
}
\def\myldots{\leaders\hbox to5pt{\hss.\hss}\hskip20pt\relax}
```

If someone want to print ⟨*text*⟩ before ⟨*word*⟩ or ⟨*braces*⟩ after ⟨*word*⟩ then he can use a control sequence \csname-⟨*word*⟩\endcsname. The example follows in the next macro \printindexentry.

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

The \printindexentry {⟨*word*⟩} macro prints an item of the ⟨*word*⟩ in the index. It starts in vertical mode inside column, prints the item and it have to switch to vertical mode back by \par command.

docby.tex

```
\def\printindexentry #1{%
     \expandafter \expandafter\expandafter \separeright \csname-#1\endcsname\end
     \apinum=\pgref[+#1]\relax
     \leavevmode\llap{\ttsmall \ifnum\apinum>-1 \apitext\fi\tmpa}%
         {\tt \ilink[@#1]{#1}\tmpb}:  {\bf\pglink\pgref[@#1]}%
         \ifnum\apinum>-1 , $\underline{\pglink\apinum}$\fi
         \dgnum=\pgref[@#1]\relax
         \undef{w:#1}\iftrue \setbox0=\hbox{}\else \setbox0=\hbox{\it\listofpages{#1}}\fi
         \ifdim\wd0=0pt \else, \unhbox0 \fi
     \hangindent=2\parindent \hangafter=1 \par
}
\def\separeright #1\right#2\end{\def\tmpa{#1}\def\tmpb{#2}}
```

The \separeright macro stores the ⟨*text*⟩ before the declared word into the \tmpa and the ⟨*braces*⟩ into the \tmpb. The control sequence \csname-⟨*word*⟩\endcsname is prepared by the \refdg macro. This sequence expands to ⟨*text*⟩\right⟨*braces*⟩. The page number with the \dg (or \dl) occurrence of the word is obtained by \pgref[@⟨*slovo*⟩] and the page number with \api occurrence is obtained by \pgref[+⟨*word*⟩]. This page number is underlined if it does exist.

4.7 The Source Code Listing

The \ifirst and \inext macros print the required part of source code. They start with \bgroup and calls the \printiabove macro. Each line is printed by \printiline {⟨*number*⟩}{⟨*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.

docby.tex

```
\ifx\pdfoutput\undefined
   \def\printiabove{\line{\leaders\specrule\hfill \kern2pt
         {\ttsmall \Brown\inputfilename}\kern2pt \specrule width\parindent}\nobreak
          \setsmallprinting}
   \def\printibelow{\vskip2pt\hrule\medskip}
   \def\specrule{\vrule height 2pt depth-1.6pt }
   \def\printiline#1#2{\noindent\ttstrut
```

```
268:       \hbox to\parindent{\hss#1:\kern.5em}{#2\par}\penalty11 }
269: \else
270:    \def\printiabove{\smallskip \setsmallprinting}
271:    \def\printibelow{\medskip}
272:    \def\printiline #1#2{\noindent
273:       \rlap{\Yellow \ttstrut width\hsize}%
274:       \ifx\isnameprinted\undefined
275:           \rlap{\line{\hss \raise8.5pt
276:              \hbox{\ttsmall \Brown \vrule height5pt width0pt \inputfilename}}}%
277:           \let\isnameprinted=\relax
278:       \fi
279:       \hbox to\parindent{\hss\Brown#1:\Black\kern.5em}{#2\par}\penalty11 }
280: \fi
```

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 275 and 276). After the `\egroup` (inserted at the end of `\ifirst` or `\inext`) the default value of `\isnameprinted` is restored. This value is `\undefined`.

4.8 The `\begtt ... \endtt` Printing

The `\begtt` establishes a new group and calls the `\printvabove` macro. Next, each printed line is processed by `\printvline` {⟨*number*⟩}{⟨*text*⟩} macro. At the end, the `\printvbelow` macro is invoked and the group is closed.

The implicit design doesn't print the numbers of lines. We draw only lines above and below in DVI mode. Moreover, we draw yellow lines in PDF mode and the yellow lines left and right in each line by `\rlap` macro.

docby.tex

```
284: \ifx\pdfoutput\undefined
285:    \def\printvabove{\smallskip\hrule\nobreak\smallskip\setnormalprinting}
286:    \def\printvbelow{\nobreak\smallskip\hrule\smallskip}
287:    \def\printvline#1#2{\hbox{\ttstrut\indent#2}\penalty12 }
288: \else
289:    \def\printvabove{\medskip\Yellow\hrule height2pt \setnormalprinting\nobreak}
290:    \def\printvbelow{\Yellow\hrule height2pt \Black\medskip}
291:    \def\printvline#1#2{\noindent
292:      \rlap{\hbox to\hsize{\Yellow\ttstrut width25pt\hfil
293:         \vrule width25pt\Black}}\hbox{\indent#2}\par\penalty12 }
294: \fi
```

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`.

docby.tex

```
298: \newdimen\figwidth  \figwidth=\hsize \advance\figwidth by-\parindent
```

DVI mode: The macro `\ifig` ⟨*width-ratio*⟩␣⟨*filename*⟩␣ inserts the picture from ⟨*filename*⟩`.eps` usig the `epsf.tex` macro package. PDF mode: The macro inserts the picture from ⟨*filename*⟩`.pdf` by pdfTEX primitive commands `\pdfximage`, `\pdfrefximage`, `\pdflastximage`.

docby.tex

```
300: \ifx\pdfoutput\undefined
301:    \input epsf
302:    \def\ifig #1 #2 {\bigskip\indent
```

\specrule: 21 \isnameprinted: 22 \printvabove: 22, 28 \printvline: 22, 28
\printvbelow: 22, 28–29 \figwidth: 22–23 \ifig: 13, 22–23

```
303:       \hbox{\epsfxsize=#1\figwidth\epsfbox{\figdir#2.eps}}\bigskip}
304: \else
305:    \def\ifig #1 #2 {\bigskip\indent
306:        \hbox{\pdfximage width#1\figwidth {\figdir#2.pdf}%
307:              \pdfrefximage\pdflastximage}\bigskip}
308: \fi
309: \def\figdir{fig/}
```

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 ⟨*mark*⟩␣ is the global variant of \item macro. The \item macro is the same as in plainTEX by default but it changes its behavior inside \begitems...\enditems environment.

docby.tex

```
313: \newcount\itemno
314: \def\begitems{\medskip\begingroup\advance\leftskip by\parindent \let\item=\dbtitem}
315: \def\dbtitem #1 {\par\advance\itemno by1 \noindent\llap{\ifx*#1$\bullet$\else#1\fi\kern3pt}}
316: \def\enditems{\medskip\endgroup}
```

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:

docby.tex

```
321: \def\dbtwarning#1{\immediate\write16{DocBy.TeX WARNING: #1.}}
```

The macros \defsec {⟨*text*⟩}, \edefsec {⟨*text*⟩} and \undef {⟨*text*⟩} define control sequence \csname⟨*text*⟩\endcsname.

docby.tex

```
323: \def\defsec#1{\expandafter\def\csname#1\endcsname}
324: \def\edefsec#1{\expandafter\edef\csname#1\endcsname}
325: \def\undef#1\iftrue{\expandafter\ifx\csname#1\endcsname\relax}
```

You can use the \undef macro in following way:

```
\undef{⟨text⟩}\iftrue ⟨sequence is undefined⟩ \else ⟨sequence is defined⟩ \fi
```

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

The \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 \obrace, \cbrace, \percent, \inchquote are defined here.

docby.tex

```
327: {\catcode`\^^I=\active \gdef^^I{\space\space\space\space\space\space\space\space}
328:  \catcode`\|=0 \catcode`\\=12 |gdef|nb{\}}
329: \bgroup
330:    \catcode`\[=1 \catcode`]=2 \catcode`\{=12 \catcode`\}=12 \catcode`\%=12
331:    \gdef\obrace[{] \gdef\cbrace[}] \gdef\percent[%]
332: \egroup
333: \def\inchquote{"}
```

The \softinput macro inputs the specified file only if this file exists. Else the warning is printed.

---

docby.tex

```
\def\softinput #1 {\let\next=\relax \openin\infile=#1
   \ifeof\infile \dbtwarning{The file #1 does not exist, run me again}
   \else \closein\infile \def\next{\input #1 }\fi
   \next}
```

The \setverb macro sets the cactodes of all special characters to normal (12).

docby.tex

```
\def\setverb{\def\do##1{\catcode`##1=12}\dospecials}
```

5.2 Initialization

DocBy.TEX prints on the terminal:

docby.tex

```
\immediate\write16{This is DocBy.TeX, version \dbtversion, modes:
    \ifx\mubyte\undefined NO\fi enc+%
    \ifx\pdfoutput\undefined DVI\else PDF\fi+%
    \ifnum\language=0 ENG\else CS\fi}
```

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

docby.tex

```
\def\dbtversion {May 2014} % version of DocBy.TeX
```

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).

docby.tex

```
\input utf8off \csname clearmubyte\endcsname
```

The encTEX mode is detected and initialized:

docby.tex

```
\ifx\mubyte\undefined % encTeX ??
   \dbtwarning{encTeX is not detected}
   \message{ \space The documented words will be not recognized in source code.}
   \message{ \space Use pdfetex -ini -enc format.ini to make
                    your format with encTeX support.}
   \csname newcount\endcsname \mubytein
   \def\enctextable#1#2{}
   \def\noactive#1{}
\else
   \def\enctextable#1#2{%
      \def\tmp ##1,#1,##2\end{\ifx^^X##2^^X}%
      \expandafter \tmp \owordbuffer ,#1,\end
         \expandafter \mubyte \csname.#1\endcsname #1\endmubyte \fi
      \expandafter \gdef \csname.#1\endcsname {#2}%
   }
   \def\noactive#1{\mubyte \emptysec ##0 #1\endmubyte}
   \def\emptysec{}
\fi
```

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

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

The \noactive {⟨text⟩} macro inserts the ⟨text⟩ as a key in the encTEX table. This key is not removed from input but the \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 \noactive then we rewrite its behavior.

\setverb: 24, 26, 28, 44 \dbtversion: 24 \enctextable: 24–25, 29, 31–32, 39
\owordbuffer: 24–25 \noactive: 6, 15, 24–25, 29, 32 \emptysec: 24

All occurrences of documented words ⟨*word*⟩ is transformed to \sword {⟨*word*⟩} by encTEX. The hyperlink is created by this macro:

docby.tex

```
370: \def\sword#1{\ilink[@#1]{#1}\write\reffile{\string\refuseword{#1}{\the\pageno}}}
```

The \onlyactive {⟨*before*⟩}{⟨*word*⟩}{⟨*post*⟩} command inserts the ⟨*word*⟩ into the list of prohibited words \owordbuffer (only if this word isn't here already). EncTEX changes all occurrences of ⟨*before*⟩⟨*word*⟩⟨*post*⟩ to \oword{⟨*before*⟩}{⟨*word*⟩}{⟨*post*⟩}. Moreover, the ⟨*word*⟩ is deactivated by \noactive (may be it was activated when \reffile is read). The \oword {⟨*before*⟩}{⟨*word*⟩}{⟨*post*⟩} command prints ⟨*before*⟩ by normal font, then runs \.⟨*word*⟩ if it is defined (else prints ⟨*word*⟩ normally). Finally, it prints ⟨*post*⟩ by normal font.

docby.tex

```
372: \def\onlyactive #1#2#3{\enctextable{#1#2#3}{\oword{#1}{#2}{#3}}%
373:    \def\tmp ##1,#2,##2\end{\ifx^^X##2^^X}%
374:    \expandafter \tmp \owordbuffer ,#2,\end
375:       \addtext #2,\to\owordbuffer \noactive{#2}\fi}
376: \def\owordbuffer{,}
377: \def\oword#1#2#3{#1\undef{.#2}\iftrue #2\else\csname.#2\endcsname\fi #3}
```

The DVI/PDF mode is initialized here:

docby.tex

```
379: \ifx\pdfoutput\undefined
380:    \dbtwarning{pdfTeX is not detected}
381:    \message{ \space The document will be without colors and hyperlinks.}
382:    \message{ \space Use pdfTeX engine, it means:  pdfetex command, for example.  }
383: \else
384:    \pdfoutput=1
385: \fi
```

5.3 The \ifirst, \inext, \ilabel Macros

The \lineno register is the number of the line, \ttlineno register is the number of the line in the \begtt...\endtt environment. We use \ifcontinue for loop controlling and \ifskipping for setting \skippingfalse and \skippingtrue.

docby.tex

```
389: \newcount\lineno
390: \newcount\ttlineno
391: \newif\ifcontinue
392: \newif\ifskipping  \skippingtrue
393: \newread\infile
```

The \ifirst {⟨*filename*⟩}{⟨*from*⟩}{⟨*to*⟩}{⟨*why*⟩} command analyses its parameter ⟨*why*⟩ by \readiparamwhy and tries to open the file ⟨*filename*⟩ for reading by \openin primitive. If it is unsuccessful then a warning is printed. Else the ⟨*filename*⟩ is stored to \inputfilename macro and other parameters are analyzed by \scaniparam. The ⟨*from*⟩ resp. ⟨*to*⟩ parameter is stored to \tmpa resp. \tmpb macro. The ⟨*num*⟩ parameter from \count=⟨*num*⟩ is stored to \tmpA and \tmbB macros. The command \insinternal is invoked with expanded parameters ⟨*from*⟩ and ⟨*to*⟩. The expansion is done via \edefed macro \act.

docby.tex

```
395: \def\ifirst #1#2#3#4{\par\readiparamwhy#4..\end
396:    \openin\infile=#1 \global\lineno=0
397:    \ifeof\infile
398:       \dbtwarning {I am not able to open the file "#1" to reading}
399:    \else
400:       \xdef\inputfilename{#1}
401:       \scaniparam #2^^X\tmpa\tmpA \scaniparam #3^^X\tmpb\tmpB
402:       {\let~=\space \def\empty{^^B^^E}\let\end=\relax \uccode`\~=`\"\uppercase{\let~}"%
403:       \noswords \xdef\act{\noexpand\insinternal {\tmpa}{\tmpb}}}\act
404:    \fi
405: }
```

\sword: 24–26, 31–32, 39 \onlyactive: 7, 24–25 \oword: 25 \lineno: 8, 25, 27–28
\ttlineno: 25, 29 \ifcontinue: 25–27, 37 \ifskipping: 25, 27–28 \skippingfalse: 8, 9, 28
\skippingtrue: 8, 25, 28 \ifirst: 7, 8–9, 14–15, 21–22, 25–26 \inputfilename: 21–22, 25–27

The \inext {⟨from⟩}{⟨to⟩}{⟨why⟩} macro does the analogical work as the \ifirst. 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.

docby.tex

```
406: \def\inext #1#2#3{\par\readiparamwhy#3..\end
407:    \ifx\inputfilename\undefined
408:       \dbtwarning {use \string\ifirst\space before using of \string\inext}
409:    \else
410:       \ifeof\infile
411:          \dbtwarning {the file "\inputfilename" is completely read}
412:       \else
413:          \scaniparam #1^^X\tmpa\tmpA \scaniparam #2^^X\tmpb\tmpB
414:          {\let~=\space \def\empty{^^B^^E}\let\end=\relax \uccode`\~=`\"\uppercase{\let~}"%
415:          \noswords \xdef\act{\noexpand\insinternal{\tmpa}{\tmpb}}}\act
416:    \fi\fi
417: }
```

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

docby.tex

```
418: \def\noswords{\def\sword##1{##1}\def\lword##1{##1}\def\fword##1##2##3{##2}%
419:    \let\flword=\fword \def\leftcomment{}\def\returntoBlack{}\def\linecomment{}}
```

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

docby.tex

```
421: \def\readiparamwhy#1#2#3\end{\let\startline=#1\relax\let\stopline=#2\relax}
```

The \scaniparam ⟨param⟩^^X⟨out⟩⟨outnum⟩ reads ⟨param⟩ in the form \count=⟨num⟩␣⟨text⟩. It stores the ⟨text⟩ to the ⟨out⟩ control sequence and ⟨num⟩ to the ⟨outnum⟩ control sequence. The prefix \count=⟨num⟩ 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=⟨num⟩ is missing then ⟨outnum⟩ is one.

docby.tex

```
423: \def\scaniparam{\futurelet\nextchar\scaniparamA}
424: \def\scaniparamA{\ifx\nextchar\count \expandafter\scaniparamB
425:    \else \def\tmp{\scaniparamB \count=1 }\expandafter\tmp
426:    \fi}
427: \def\scaniparamB \count{\afterassignment\scaniparamC\tempnum}
428: \def\scaniparamC #1^^X#2#3{\def#2{#1}\edef#3{\the\tempnum}}
```

The main work (inserting of source code) is done by the macro \insinternal with parameters {⟨from⟩}{⟨to⟩}.

docby.tex

```
430: \def\insinternal #1#2{%
431:    \bgroup
432:       \printiabove  % graficke zpracovani zacatku
433:       \setverb \catcode`\"=12 \catcode`\^^M=9 \catcode`\^^I=\active
434:       \mubytein=1 \obeyspaces \continuetrue \tempnum=\tmpA\relax
435:       \def\testline##1#1##2\end{\ifx^^Y##2^^Y\else \nocontinue \fi}%
436:       \ifx^^X#1^^X\def\testline##1\end{\nocontinue}\fi
437:       \loop % preskakovani radku
438:          \ifeof\infile \returninsinternal{Text "#1" not found (\string\count=\the\tempnum)}{}\fi
439:          \readnewline
440:          \expandafter \testline \expandafter^^B\etext ^^E#1\end
441:          \ifcontinue \repeat
442:       \let\lastline=\empty
443:       \continuetrue  \tempnum=\tmpB\relax
444:       \def\testline##1#2##2\end{\ifx^^Y##2^^Y\else \nocontinue \fi}%
445:       \ifx^^X#2^^X\def\testline##1\end{\nocontinue}\fi
446:       \ifx+\startline \printilineA
447:          \expandafter \testline \expandafter ^^B\etext ^^E#2\end
448:          \ifcontinue\else\returninsinternal{}\fi
```

```
449:          \readnewline
450:       \else
451:          \readnewline
452:          \ifskipping\ifx\text\empty \readnewline \fi\fi
453:       \fi
454:       \loop  % pretisk radku
455:          \expandafter \testline \expandafter ^^B\etext ^^E#2\end
456:          \ifcontinue
457:             \printilineA
458:             \ifeof\infile \returninsinternal{}\fi
459:             \readnewline \repeat
460:       \ifx+\stopline \printilineA
461:          \ifx\lastline\relax \else \printiline{\lastline}{}\relax\fi
462:       \fi
463:       \global\let\Black=\oriBlack % pokud jsme skoncili vypis uvnitr komentare
464:       \printibelow  % graficke zpracovani konce
465:    \egroup\gdef\ilabellist{}\Black
466: }
```

The `\isinternal` macro has two main loops. First one (from line 437 to 441) 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 454 to 459) 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.

docby.tex

```
467: \def\nocontinue{\advance\tempnum by-1 \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+\startline` is a test if user want to print the first line. The `\ifx+\stopline` 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 464. The inserted conditionals have to by 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.

docby.tex

```
469: \def\returninsinternal #1#2#3\printibelow{%
470:    \ifx^^X#1^^X\else
471:      \dbtwarning{#1 in file \inputfilename}\fi
472:    #2\fi\printibelow
473: }
```

The `\readnewline` is simple:

docby.tex

```
474: \def\readnewline {\read\infile to\text \global\advance\lineno by1\relax
475:    {\noswords \xdef\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.

\testline: 26–27 \nocontinue: 26–27 \returninsinternal: 26–27 \readnewline: 26–27

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).

docby.tex

```
\def\printilineA {%
   \ifskipping\else \ifx\text\empty \def\text{ }\fi\fi % trik pro pripad \skippingfalse
   \ifx\text\empty
      \ifx\lastline\empty % nacten prvni prazdny radek
         \let\lastline=\relax
      \else               % nacten pozdejsi prazdny radek
         \ifx\lastline\relax \else \printiline{\lastline}{}\relax\fi
         \edef\lastline{\the\lineno}%
      \fi
   \else                  % nacten plny radek
      \ifx\lastline\empty \let\lastline=\relax \fi
      \ifx\lastline\relax \else \printiline{\lastline}{}\relax\fi
      \printiline{\the\lineno}{\text}\relax
      \let\lastline=\relax
   \fi \ilabellist
}
```

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

docby.tex

```
\def\ilabellist {}
\def\ilabel [#1]#2{{\noswords\edef\act{\noexpand\ilabelee{#1}{#2}}\expandafter}\act}
\def\ilabelee #1#2{\expandafter\def\expandafter\ilabellist\expandafter{%
    \ilabellist \expandafter\testilabel\etext\end{#1}{#2}}
}
```

The \ilabel macro first expands its parameters (by the \act macro) and calls the internal \ilabelee macro. This macro adds the following text to the \ilabellist:

```
\expandafter\testilabel\etext\end{⟨label⟩}{⟨text⟩}
```

The \testilabel ⟨*line*⟩\end{⟨*label*⟩}{⟨*text*⟩} command defines the temporary \tmp macro with the ⟨*text*⟩ separator in order to test if the ⟨*text*⟩ is included in ⟨*line*⟩. If it is true then the aim of the reference is registered by the \labeltext command.

docby.tex

```
\def\testilabel#1\end#2#3{%
    \def\tmp ##1#3##2\end{\ifx^^Y##2^^Y\else
       \undef{d:#2}\iftrue \defsec{d:#2}{}\labeltext[#2]{\the\lineno}\fi\fi}
    \tmp^^B#1^^E#3\end
}
```

5.4 Commands \begtt, \endtt

The \begtt and \endtt macros are described in "TEXbook inside out" (the book in Czech language) in pages 27–30. The \startverb macro reads the following text separated by the word \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.

docby.tex

```
\def\begtt {\bgroup\printvabove
   \setverb \catcode`\"=12 \catcode`\^^M=12 \obeyspaces
   \begtthook\relax \startverb}
{\catcode`\|=0 \catcode`\^^M=12 \catcode`\\=12 %
```

\printilineA: 26–28 \lastline: 26–28 \ilabellist: 27–28 \ilabel: 9, 27–28 \ilabelee: 28
\testilabel: 28 \begtt: 9, 14, 22, 25, 28–29 \startverb: 28 \runttloop \endttloop: 29
\scannexttoken: 29

```
510:   |gdef|startverb^^M#1\endtt{|runttloop#1|end^^M}%
511:   |gdef|runttloop#1^^M{|ifx|end#1 |expandafter|endttloop%
512:      |else|global|advance|ttlineno by1 %
513:        |printvline{|the|ttlineno}{#1}|relax|expandafter|runttloop|fi}} %
514: \def\endttloop#1{\printvbelow\egroup\futurelet\nextchar\scannexttoken}
515: \long\def\scannexttoken{\ifx\nextchar\par\else\noindent\fi}
```

The number of line globally incremented in \begtt...\endtt is stored in \ttlineno register. You can set this register to zero at each begin of section (for example).

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.

docby.tex

```
519: \def\namespacemacro#1{}
```

We need to set a label to each name space. The label is the text expanded by \namespacemacro{@!} 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 have to know all local words declared in it in order to the 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.

docby.tex

```
521: \def\namespace #1{%
522:    \let\tmp=\namespacemacro
523:    \gdef\namespacemacro##1{#1}%
524:    \global\expandafter\let\csname no:\namespacemacro{@!}\endcsname\tmp
525:    \ewrite{\string\refns{\namespacemacro{@!}}}%
526:    \def\locword##1{%
527:       \global\expandafter\let
528:          \csname\namespacemacro{@!},##1\expandafter\endcsname\csname.##1\endcsname
529:       \enctextable{##1}{\lword{##1}}}%
530:    \csname ns:\namespacemacro{@!}\endcsname
531: }
```

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 to 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.

docby.tex

```
532: \def\endnamespace{\if^^X\namespacemacro{@!}^^X\else
533:    \def\locword##1{%
534:       \global\expandafter\let
535:          \csname.##1\expandafter\endcsname\csname\namespacemacro{@!},##1\endcsname
536:       \undef{.##1}\iftrue \noactive{##1}\fi}%
537:    \csname ns:\namespacemacro{@!}\endcsname
```

\namespacemacro: 29–30 \namespace: 11, 15, 29 \locword: 29, 39 \endnamespace: 11, 15, 29

```
538:    \ewrite{\string\refnsend{\namespacemacro{@!}}}%
539:    \global\expandafter\let\expandafter\namespacemacro\csname no:\namespacemacro{@!}\endcsname
540:    \fi
541: }
```

These macros used the \ewrite sequence which writes the text to the \reffile with delay (in output routine) but the expansion is done immediately. But the \nb control sequence is not expanded.

docby.tex

```
542: \def\ewrite#1{{\let\nb=\relax \edef\act{\write\reffile{#1}}\act}}
```

EncTEX stores the control sequence \.⟨*word*⟩ instead local ⟨*word*⟩ at the start of each name space. The \.⟨*word*⟩ macro expands to \lword{⟨*word*⟩}. If the local word occurs the \lword works as follows:

docby.tex

```
544: \def\lword#1{\genlongword\tmp{#1}\ilink[@\tmp]{#1}%
545:     \ewrite{\string\refuseword{\tmp}{\noexpand\the\pageno}}}
546: \def\genlongword#1#2{\expandafter\def\expandafter#1\expandafter{\namespacemacro{#2}}}
```

The \genlongword ⟨*tmp*⟩{⟨*word*⟩} command creates long version of the ⟨*word*⟩ from short variant of it and stores this log version to ⟨*tmp*⟩ macro. The occurrence of the ⟨*word*⟩ is presented by the parameter of the \ilink command and by the long name (unambiguous) written to the \reffile. The short variant of the word is printed.

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

docby.tex

```
548: \def\refns#1{\edefsec{o:#1}{\currns}
549:    \edef\currns{#1}\undef{ns:\currns}\iftrue \defsec{ns:\currns}{}\fi}
550: \def\refnsend#1{\edef\currns{\csname o:#1\endcsname}}
551: \def\currns{}
```

The \refns macro remembers the previous ⟨*nslabel*⟩ which is stored in \currns. This value is stored to the \o:⟨*new-nslabel*⟩ and the \currns is redefined as ⟨*new-nslabel*⟩. The implicit value of the \ns:⟨*nslabel*⟩ is empty. The \refdg commands add information to the \ns:⟨*nslabel*⟩ buffer (see lines 850–853 in section 5.9). Finally, the \refnsend command returns the \currns macro to the original value before name space was started.

5.6 The \dg Command and Friends

The macros \dg, \dl, \dgn, \dgh, \dln, \dlh save its name to a \tmpA and then they scan parameters by a \dgpar. Finally they run the internal version for itself \csname␣ii\tmpA\endcsname.

docby.tex

```
555: \def\dg{\def\tmpA{dg}\dgpar}  \def\dgn{\def\tmpA{dgn}\dgpar}  \def\dgh{\def\tmpA{dgh}\dgpar}
556: \def\dl{\def\tmpA{dl}\dgpar}  \def\dln{\def\tmpA{dln}\dgpar}  \def\dlh{\def\tmpA{dlh}\dgpar}
557:
558: \def\dgpar {\futurelet\nextchar\dgparA}
559: \def\dgparA {\ifx\nextchar[\def\tmp{\dparam}\else\def\tmp{\dparam[]}\fi\tmp}
```

The previous macros prepare the reading of optional parameter. The main work is done by the \dparam macro.

docby.tex

```
561: \def\dparam [#1]#2 {%
562:     \def\printbrackets{}%
563:     \ifx^^X#2^^X\nextdparam{#1}\fi
564:     \def\tmpa{#2}\def\tmpb{}%
565:     \varparam,\tmpa, \varparam.\tmpa.  \varparam;\tmpa; \varparam:\tmpa:
566:     \expandafter\managebrackets\tmpa()\end
567:     {\let\nb=\relax
568:     \edef\act{\expandafter\noexpand \csname ii\tmpA\endcsname{#1}{\tmpa}{\printbrackets}}%
```

\ewrite: 29–30, 32, 35 \lword: 26, 29–30 \genlongword: 30, 32 \refns: 29–30, 38
\refnsend: 30, 38 \currns: 30, 39 \dg: 9, 6–7, 10–13, 20–21, 24, 30–32, 38–39
\dl: 9, 10–11, 13, 20–21, 29–32, 38–39 \dgn: 9, 10, 13, 30 \dgh: 9, 10, 13, 30 \dln: 9, 10, 13, 30
\dlh: 9, 10, 13, 30 \dgpar: 30 \dparam: 30–31

```
569:     \expandafter}\act
570:     \tmpb \if|\expandafter\ignoretwo\tmpA|\expandafter\maybespace\fi
571: }
572: \def\nextdparam#1#2\maybespace\fi{\fi\dparam[#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 ⟨*word*⟩) 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`:

docby.tex

```
574: \def\varparam#1{\def\tmp ##1#1##2 {\def\tmpa{##1}\if^^X##2^^X\else
575:         \expandafter\gobblelast\tmpb\end#1##2\fi}%
576:    \expandafter\tmp}
577: \def\gobblelast#1\end#2{\def\tmp##1#2{\def\tmpb{#2##1#1}}\tmp}
```

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

At the end of this work, we have the ⟨*word*⟩ in `\tmpa` 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.

docby.tex

```
579: \def\managebrackets #1()#2\end{\def\tmpa{#1}%
580:    \if|#2|\else\def\printbrackets{()}\fi}
```

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.

docby.tex

```
582: \def\maybespace{\futurelet\tmp\domaybespace}
583: \def\domaybespace{\let\next=\space
584:    \ifx\tmp‘\def\next##1{}\fi
585:    \next}
```

The `\dparam` macro changes the original command `\dg*`, `\dl*` respectively to internal variant `\iidg*`, `\iidl*` respectively. This is done on the line 568. 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 encTEX table (this is redundant because the same work is done when `\reffile` is read by `\refdg` macro). Next, the `\iidg` macro creates the aim of the reference in the form `@`⟨*word*⟩ and saves `\refdg{`⟨*text*⟩`}{`⟨*word*⟩`}{`⟨*brackets*⟩`}{}` to the `\reffile`. The highlighted ⟨*word*⟩ is printed by the `\printdg` command and the footnote is inserted by the `\printfnote` command.

docby.tex

```
587: \def\iidg #1#2#3{%
588:    \enctextable{#2}{\sword{#2}}%
589:    \label [@#2]%
590:    \write\reffile{\string\refdg{#1}{#2}{#3}{}}%
591:    \printdg{#1}{#2}{#3}%
592:    \printfnote{#1}{#2}{#3}{#2}%
593: }
```

The `\iidl` creates the aim of the reference by `\label`␣`[@`⟨*long-word*⟩`]`, writes the information to the `\reffile` in the format `\refdg{`⟨*text*⟩`}{`⟨*long-word*⟩`}{`⟨*brackets*⟩`}{`⟨*short-word*⟩`}` and prints the ⟨*short-word*⟩ highlighted by `\printdg` command. It stores nothing to the encTEX table. Finally, it inserts the footnote by `\printfnote{`⟨*text*⟩`}{`⟨*long-word*⟩`}{`⟨*braces*⟩`}`.

\nextdparam: 30–31 \varparam: 30–31 \gobblelast: 31 \managebrackets: 30–31
\printbrackets: 30–31 \maybespace: 31 \iidg: 10, 20, 31–32 \iidl: 10, 31–32

docby.tex
```
\def\iidl #1#2#3{%
    \genlongword\tmpB{#2}%
    \ifx\tmpB\empty \dbtwarning{\string\dl\space#2 outside namespace, ignored}%
    \else
       \expandafter\label\expandafter [\expandafter @\tmpB]%
       \ewrite{\string\refdg{#1}{\tmpB}{#3}{#2}}%
       \printdg{#1}{#2}{#3}%
       \printfnote{#1}{\tmpB}{#3}{#2}%
    \fi
}
```

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.

docby.tex
```
\def\iidgh#1#2#3{{\def\printdg##1##2##3{}\iidg{#1}{#2}{#3}}}
\def\iidlh#1#2#3{{\def\printdg##1##2##3{}\iidl{#1}{#2}{#3}}}
```

The \iidgn command redefines the \.⟨word⟩ macro which is inserted to the text by encTEX. The result of the expansion will be \fword{⟨text⟩}{⟨word⟩}{⟨braces⟩} instead of the common result \sword{⟨word⟩}.

docby.tex
```
\def\iidgn#1#2#3{\enctextable{#2}{\fword{#1}{#2}{#3}}}
```

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

docby.tex
```
\def\fword#1#2#3{\iidgh{#1}{#2}{#3}\printdginside{#2}{#2}}
```

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⟩}{⟨braces⟩}.

docby.tex
```
\def\iidln#1#2#3{%
    \global\expandafter\let\csname;#2\expandafter\endcsname\csname.#2\endcsname
    \enctextable{#2}{\flword{#1}{#2}{#3}}}
```

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{⟨word⟩} because there is no possibility to remove the item from encTEX table.

docby.tex
```
\def\flword#1#2#3{\iidlh{#1}{#2}{#3}\printdginside{#2}{#2}%
    \global\expandafter\let\csname.#2\expandafter\endcsname\csname;#2\endcsname
    \undef{.#2}\iftrue \noactive{#2}\fi}
```

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:

docby.tex
```
\let\footnote=\undefined
```

Our special footnotes use the "insert" \footins declared in plainTEX. 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 pagenumber lists in the footnotes but they are not definite because of new pagebreaks. The new pagebreaks influence new lists of pagenumbers 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 hight (10pt) multiplied by this coefficient. I need to set the \count\footins only.

\iidgh: 10, 32 \iidlh: 10, 32 \iidgn: 10, 32 \fword: 26, 32 \iidln: 10, 32
\flword: 26, 32

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 `\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 `\skip\footins` and we need to use the vertical glue above and below the footnote rule.

docby.tex

```
623: \skip\footins=18pt
624: \dimen\footins=\vsize
625: \count\footins=200
```

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

docby.tex

```
627: \newcount\totalfoocount
628: \newdimen\totalfoodim
```

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

docby.tex

```
630: \def\specfootnote#1{\insert\footins\bgroup
631:     \let\tt=\ttsmall \rmsmall
632:     \floatingpenalty=20000 \setbox0=\hbox{#1}%
633:     \ht0=10pt \dp0=0pt \box0 \egroup
634:     \global\advance\totalfoocount by1
635: }
```

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

docby.tex

```
637: \def\footnoterule \unvbox\footins {
638:    \vskip-12pt \vfil
639:    \moveright\parindent\vbox{\hsize=\nwidth \hrule
640:    \setbox2=\vbox{\unvbox\footins \unskip
641:        \setbox2=\lastbox
642:        \global\setbox4=\hbox{\unhbox2}
643:        \loop \unskip\unskip\unpenalty
644:            \setbox2=\lastbox
645:            \ifhbox2 \global\setbox4=
646:               \hbox{\unhbox2 \penalty-300\hskip15pt plus5pt \unhbox4}
647:            \repeat}
648:        \setbox2=\vbox{\hbox{} \parskip=0pt
649:        \lineskiplimit=0pt \baselineskip=10pt \raggedright \rightskip=0pt plus7em
650:        \leftskip=0pt \hyphenpenalty=10000 \noindent \Black \unhbox4 }
651:        \global\advance\totalfoodim by\ht2 \unvbox2}
652: }
```

This macro decomposes the vertical list of inserts `\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 `\endgraf`. This means that the footnotes are divided to lines. The \totalfoodim is advanced here too.

The \bye macro (see the line 790) writes the \totalfoocount and \totalfoodim to the \reffile. The actual average coefficient is added here too. This information is written only if the \indexbuffer is not empty, it means that (at least) the second TEX run is in progress.

This information is read by the \refcoef {⟨*coef*⟩}{⟨*number*⟩}{⟨*height*⟩} macro at the start of the next TEX run. It sets the average coefficient `\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 \gobblerest removes the digits after decimal point including the text `pt`.

\totalfoocount: 33, 37 \totalfoodim: 33, 37 \specfootnote: 20, 33 \refcoef: 34, 37–38
\gobblerest: 34, 41

docby.tex

```
654: \def\refcoef#1#2#3{%
655:    \ifnum#1=200 % jsme na zacatku tretiho pruchodu
656:       \dimen0=#3 \divide\dimen0 by #2
657:       \multiply \dimen0 by100
658:       \afterassignment\gobblerest \count\footins=\the\dimen0 \end
659:    \else \count\footins=#1
660:    \fi
661:    \message{foot-coef:  \the\count\footins}
662: }
663: \def\gobblerest #1\end{}
```

We need to suppress the expansion of some macros in output routine which are presented in `\write` parameter. These macros are set to `\relax` meaning in output routine. In order to the headline printing is done correctly we need to expand `\makehedaline` before the setting of these macros to relax and we need to store the result of `\makeheadline` in a box.

docby.tex

```
665: \output={\setbox0=\makeheadline \def\makeheadline{\box0\nointerlineskip}
666:   \let~=\relax \let\nb=\relax \let\TeX=\relax \let\docbytex=\relax \let\_=\relax \let\tt=\relax
667:   \outputhook \plainoutput }
```

5.8 Section, Subsection, Part

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

docby.tex

```
672: \newcount\secnum
673: \newcount\subsecnum
674: \newtoks\sectitle
675: \newif\ifsavetoc \savetoctrue
```

There is an optional parameter `[`⟨*label*⟩`]` followed by optional (ignored) space when \sec and \subsec macros are used. The last token of ⟨*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 \tmpA and the parameter scanning process is started by \secparam.

docby.tex

```
677: \def\sec{\def\tmpA{sec}\futurelet\nextchar\secparam}
678: \def\subsec{\def\tmpA{subsec}\futurelet\nextchar\secparam}
```

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

docby.tex

```
680: \def\secparam{\ifx\nextchar[%
681:      \def\tmp[##1]{\def\seclabel{##1}\futurelet\nextchar\secparamA}%
682:      \expandafter\tmp
683:    \else \def\seclabel{}\expandafter\secparamB\fi
684: }
685: \def\secparamA{\expandafter\ifx\space\nextchar
686:       \def\tmp{\afterassignment\secparamB\let\next= }\expandafter\tmp
687:    \else \expandafter\secparamB \fi
688: }
689: \def\secparamB #1\par{\nolastspace #1^^X ^^X\end}
690: \def\nolastspace #1 ^^X#2\end{\ifx^^X#2^^X\secparamC #1\else \secparamC #1^^X\fi}
691: \def\secparamC #1^^X{\sectitle={#1}\csname ii\tmpA\endcsname}
```

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

\secnum: 18, 34–36 \subsecnum: 18, 34–35 \sectitle: 18–19, 34–35 \ifsavetoc: 18, 34–35
\sec: 12, 15, 17–18, 34, 39 \subsec: 12, 18, 34 \tmpA: 18, 25–26, 30–31, 34–35
\secparam: 18, 34–35 \seclabel: 18, 34–35 \secparamA: 34 \secparamB: 34 \nolastspace: 34
\setparamC \iisec: 34–35 \makelinks: 17–18, 35

docby.tex

```
\def\iisec{%
    \ifsavetoc \global\advance\secnum by1 \global\subsecnum=0 \fi
    \edef\makelinks{%
        \ifsavetoc \noexpand\savelink[sec:\the\secnum]\fi
        \if^^X\seclabel^^X\else \noexpand\labeltext[\seclabel]{\the\secnum}\fi}
    \expandafter \printsec \expandafter{\the\sectitle}% vlozi horni mezeru, text, nakonec \par
    \ifsavetoc
       \ewrite {\string\reftocline{\the\secnum}{\the\sectitle}{\noexpand\the\pageno}}\fi
    \mark{\ifsavetoc \the\secnum\space \else
          \ifx\emptynumber\empty\else\emptynumber\space\fi\fi \the\sectitle}
    \savetoctrue \printsecbelow
}
```

The \iisubsec macro is similar as \iisec.

docby.tex

```
\def\iisubsec {%
    \ifsavetoc \global\advance\subsecnum by1 \fi
    \edef\makelinks{%
        \ifsavetoc \noexpand\savelink[sec:\the\secnum.\the\subsecnum]\fi
        \if^^X\seclabel^^X\else \noexpand\labeltext[\seclabel]{\the\secnum.\the\subsecnum}\fi}
    \expandafter \printsubsec \expandafter{\the\sectitle}% vlozi horni mezeru, text, nakonec \par
    \ifsavetoc  \ewrite
         {\string\reftocline{\the\secnum.\the\subsecnum}{\the\sectitle}{\noexpand\the\pageno}}\fi
    \savetoctrue \printsubsecbelow
}
```

The \part macro uses the conversion of the \partnum register to letters. It is implemented as \thepart macro.

docby.tex

```
\newcount\partnum
\def\thepart{\ifcase\partnum --\or A\or B\or C\or D\or E\or F\or G\or
  H\or I\or J\or K\or L\or M\or N\or O\or P\or Q\or R\or S\or T\or
  U\or V\or W\or X\or Y\or Z\else +\the\partnum\fi}
```

The \part macro is implemented by \iipart and it is similar to \iisec.

docby.tex

```
\def\part{\def\tmpA{part}\futurelet\nextchar\secparam}
\def\iipart{%
    \ifsavetoc \global\advance\partnum by1 \fi
    \edef\makelinks{%
        \ifsavetoc \noexpand\savelink[sec:\thepart]\fi
        \if^^X\seclabel^^X\else \noexpand\labeltext[\seclabel]{\thepart}\fi}
    \expandafter \printpart \expandafter{\the\sectitle}% vlozi horni mezeru, text, nakonec \par
    \ifsavetoc
       \ewrite {\string\reftocline{}{\the\sectitle}{\noexpand\the\pageno}}\fi
    \savetoctrue \printpartbelow
}
```

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 baselineskip. The \ilink (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.

docby.tex

```
\ifx\pdfoutput\undefined
   \def\savelink[#1]{}
   \def\ilink [#1]#2{#2}
   \def\savepglink{}
   \def\pglink{\afterassignment\dopglink\tempnum=}
   \def\dopglink{\the\tempnum}
   \def\ulink[#1]#2{#2}
\else
   \def\savelink[#1]{\ifvmode\nointerlineskip\fi
```

\iisubsec: 34–35 \partnum: 21, 35, 40 \thepart: 18, 21, 35, 40 \part: 12, 14, 35 \iipart: 35
\savelink: 35–36 \iilink \linkskip: 18, 36

```
746:       \vbox to0pt{\def\nb{/_}\vss\pdfdest name{#1} xyz\vskip\linkskip}}
747:    \def\ilink [#1]#2{{\def\nb{/_}\pdfstartlink height 9pt depth 3pt
748:       attr{/Border[0 0 0]} goto name{#1}}\Blue#2\Black\pdfendlink}
749:    \def\savepglink{\ifnum\pageno=1 \pdfdest name{sec::start} xyz\relax\fi % viz \bookmarks
750:        \pdfdest num\pageno fitv\relax}
751:    \def\pglink{\afterassignment\dopglink\tempnum=}
752:    \def\dopglink{\pdfstartlink height 9pt depth 3pt
753:        attr{/Border[0 0 0]} goto num\tempnum\relax\Blue\the\tempnum\Black\pdfendlink}
754:    \def\ulink[#1]#2{\pdfstartlink height 9pt depth 3pt
755:    user{/Border[0 0 0]/Subtype/Link/A << /Type/Action/S/URI/URI(#1)>>}\relax
756:    \Green{\tt #2}\Black\pdfendlink}
757: \fi
758: \newdimen\linkskip \linkskip=12pt
```

These macros have special implementation for DVI and PDF modes. The blue color for links are declared in the `\ilink` 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 unexpanded 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 page ⟨*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 `\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.

docby.tex

```
760: \def\reflabel #1#2#3{%
761:    \undef{^^Y#1}\iftrue
762:       \ifx^^X#2^^X\else\defsec{^^X#1}{#2}\fi
763:       \defsec{^^Y#1}{#3}%
764:    \else
765:       \dbtwarning{The label [#1] is declared twice}%
766:    \fi
767: }
768: \def\numref [#1]{\undef{^^X#1}\iftrue \else \csname^^X#1\endcsname\fi}
769: \def\pgref [#1]{\undef{^^Y#1}\iftrue-1000\else \csname^^Y#1\endcsname\fi}
```

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*⟩`}` us 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 ⟨*label*⟩ parameter cannot be expanded) and the auxiliary macro `\writelabelinternal` `{`⟨*text*⟩`}{`⟨*label*⟩`}` is used. The first part, i.e. `\writelabel{`⟨*text*⟩`}` is expanded by `\edef`.

docby.tex

```
771: \def\labeltext[#1]#2{\savelink[#1]\writelabel[#1]{#2}}
772: \def\writelabel[#1]#2{\edef\tmp{\noexpand\writelabelinternal{#2}}\tmp{#1}}
773: \def\writelabelinternal#1#2{\write\reffile{\string\reflabel{#2}{#1}{\the\pageno}}}
```

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

docby.tex

```
775: \def\label[#1]{\labeltext[#1]{}}
```

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.

\savepglink: 19, 35–36, 40 \pglink: 20–21, 35–36, 43 \dopglink: 35–36 \reflabel: 36–38
\numref: 12, 13, 36–37 \pgref: 12, 13, 20–21, 36–37 \labeltext: 13, 28, 35–36 \writelabel: 36
\writelabelinternal: 36 \label: 12, 13, 31–32, 36–37 \cite: 12, 9, 11, 13, 37

docby.tex

```
\def\cite[#1]{\ifnum \pgref[#1]=-1000
      \dbtwarning{label [#1] is undeclared}\Red??\Black
   \else \edef\tmp{\numref[#1]}%
      \ifx\tmp\empty \edef\tmp{\pgref[#1]}\fi
      \ilink[#1]{\tmp}%
   \fi
}
```

The links are solved in \api {⟨word⟩} too. This macro uses \label[+⟨word⟩] and saves the ⟨word⟩ prefixed by \refapiword to \reffile.

docby.tex

```
\def\api #1{\label[+#1]\write\reffile{\string\refapiword{#1}}}
\def\apitext{$\succ$}
```

The \apitext is printed alongside the ⟨word⟩ in the table of contents and the index.

When the \bye is executed, the information for \refcoef (line 790) is stored in \reffile and the test of \reffile data consistence is processed.

docby.tex

```
\def\bye{\par\vfill\supereject
   \ifx\indexbuffer\empty \else % jsme ve druhem a dalsim pruchodu
      \immediate\write\reffile{\string\refcoef
         {\the\count\footins}{\the\totalfoocount}{\the\totalfoodim}}
      \immediate\closeout\reffile
      \setrefchecking \continuetrue \input \jobname.ref
      \ifcontinue  \indexbuffer \relax \fi
      \ifcontinue \ifx\text\tocbuffer \else
          \continuefalse \dbtwarning{toc-references are inconsistent, run me again}\fi
      \fi
      \ifcontinue \immediate\write16{DocBy.TeX: OK, all references are consistent.}\fi
   \fi
   \end
}
```

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.

docby.tex

```
\def\setrefchecking{\catcode`\"=12
   \def\refcoef##1##2##3{}
   \def\reflabel##1##2##3{\def\tmp{##3}\let\next=\relax
       \expandafter\ifx\csname^^Y##1\endcsname \tmp
          \ifx^^X##2^^X\else
             \def\tmp{##2} \expandafter \ifx \csname^^X##1\endcsname \tmp \else
                \continuefalse
                \dbtwarning{text references are inconsistent, run me again}
                \let\next=\endinput
          \fi\fi
       \else
          \continuefalse
          \dbtwarning{page references are inconsistent, run me again}
          \let\next=\endinput
       \fi\next}
   \def\refuseword##1##2{\expandafter \ifx\csname -##1\endcsname \relax
      \defsec{-##1}{##2}\else \edefsec{-##1}{\csname -##1\endcsname,##2}\fi}
   \def\refdg##1##2##3##4{\addtext\ptocentry @{##2}{##4}\to\tocbuffer}
   \let\text=\tocbuffer \def\tocbuffer{}
   \def\,##1{\let##1=\relax}\indexbuffer
   \def\,##1{\edef\tmp{\expandafter\ignoretwo \string ##1}%
       \expandafter\ifx \csname w:\tmp\endcsname ##1\else
          \continuefalse
          \dbtwarning{auto-references are inconsistent, run me again}
```

```
826:          \expandafter\ignoretorelax \fi}
827: }
828: \def\ignoretorelax #1\relax{}
```

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 sequence are used foe another purpose than during normal processing. First, these sequences take the \relax meaning at line 804. Second, the \, is redefined in order to do the test of equivalence of the w:⟨*word*⟩ and -⟨*word*⟩ sequences. The test is executed by \indexbuffer\relax at line 789. If an inconsistency occurs then the message is printed and macro processing is skipped to \ignoretorelax. next, the \refdgmacro 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 795.

5.10 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*⟩.

docby.tex

```
832: \def\tocbuffer{}
833: \def\indexbuffer{}
834: \def\addtext #1\to#2{\expandafter\gdef\expandafter#2\expandafter{#2#1}}
```

The following commands are used in the \reffile.

```
\reftocline{⟨number⟩}{⟨title⟩}{⟨page⟩}  % about section, subsection for TOC
\refdg{⟨before⟩}{⟨word⟩}{⟨after⟩}{⟨k-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
\refns{⟨nslabel⟩}                    % see section 5.5, name spaces
\refnsend{⟨nslabel⟩}                 % see section 5.5, name spaces
```

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

docby.tex

```
836: \def\reftocline#1#2#3{\def\currb{#1}%
837:    \istocsec#1.\iftrue \def\currsecb{#1}\else \addbookmark\currsecb \fi
838:    \addtext\dotocline{#1}{#2}{#3}\to\tocbuffer}
```

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.

docby.tex

```
840: \def\dotocline#1#2#3{\par
841:    \istocsec#1.\iftrue \ptocline{#1}{#2}{#3}\else \ptocsubline{#1}{#2}{#3}\fi}
842: \def\istocsec#1.#2\iftrue{\if^^X#2^^X}
```

The \tocbuffer includes TOC information about sections and subsections. Moreover, it includes the data about documented words stored by \refdg a \refapiword.

docby.tex

```
844: \def\refdg#1#2#3#4{%
845:     \edefsec{-#2}{#1\noexpand\right\if!#4!#3\fi}
```

\ignoretorelax: 38 \addtext: 25, 37–39, 41 \reffile: 25, 29–31, 33–34, 36–38, 40, 42
\reftocline: 34–35, 38, 40 \tocbuffer: 37–40 \dotocline: 38, 40 \istocsec: 38
\refdg: 21, 30–32, 37–40 \refapiword: 37–39

```
846:     \expandafter\addtext\csname-#2\endcsname,\to\indexbuffer
847:     \addbookmark\currb
848:     \addtext\ptocentry @{#2}{#4}\to\tocbuffer
849:     \ifx^^X#4^^X\enctextable{#2}{\sword{#2}}    % slovo je z \dg
850:     \else \expandafter\def\csname ns:\currns    % slovo je z \dl
851:           \expandafter\expandafter\expandafter\endcsname
852:           \expandafter\expandafter\expandafter
853:            {\csname ns:\currns\endcsname \locword{#4}}
854:     \fi
855: }
856: \def\refapiword#1{\addbookmark\currb \addtext\ptocentry +{#1}{}\to\tocbuffer}
```

The \refdg macro has {⟨before⟩}{⟨word⟩}{⟨after⟩}{⟨k-word⟩} parameters where ⟨before⟩ is a text before word, ⟨word⟩ is a long variant of the word and ⟨after⟩ can include optional braces {}. If the long word differ from short word (when \dl is used) then ⟨k-word⟩ includes the short variant of the word else ⟨k-word⟩ is empty. The \refdg macro stores its information to \tocbuffer and \indexbuffer in parallel. If ⟨k-word⟩ is empty then \sword is stored to encTEX table. If ⟨k-word⟩ is nonempty then namespaces are taken into account. The TOC is created by the \dotoc macro.

docby.tex

```
858: \def\dotoc{\bgroup \savetocfalse \sec \tittoc \par \smallskip
859:    \leftskip=\parindent \rightskip=\parindent plus .5\hsize
860:    \tochook \tocbuffer \par\egroup}
```

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

```
before sorting:  \-⟨word1⟩ , \-⟨word2⟩ , \-⟨word3⟩ , \-⟨word4⟩ , ...
after sorting:   \, \-⟨wordA⟩ \, \-⟨wordB⟩ \, \-⟨wordC⟩ \, \-⟨wordD⟩ ...
```

The \-⟨word⟩ means one control sequence here. Each control sequence is a macro with the body ⟨before⟩\right⟨after⟩, see the 845 line. The index is printed by the \doindex macro.

docby.tex

```
862: \def\doindex {\par\penalty0
863:    \calculatedimone
864:    \ifdim\dimen1<7\baselineskip \vfil\break \fi
865:    \sec \titindex \par
866:    \ifx\indexbuffer\empty
867:        \dbtwarning {index is empty, run me again}
868:    \else
869:        \message{DocBy.TeX: sorting index...}
870:        \sortindex
871:        \indexhook
872:        \vskip-\baselineskip
873:        \begmulti 2 \rightskip=0pt plus5em \parfillskip=0pt plus2em
874:          \widowpenalty=0 \clubpenalty=0
875:          \let\,=\doindexentry \indexbuffer \endmulti
876:    \fi
877: }
```

The \calculatedimone command and the test of \dimen1 value prepares the two columns typesetting, see the section 5.13. The \doindex begins the \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 \begmulti␣2 and the \, separator takes the meaning \doindexentry. This macro prints each index entry when \indexbuffer expands.

docby.tex

```
878: \def\doindexentry #1{%
879:       \edef\tmp{\expandafter\ignoretwo \string #1}%
880:       \expandafter \remakebackslash \tmp\end
881:       \expandafter \printindexentry \expandafter {\tmp}%
882: }
883: \def\remakebackslash#1#2\end{\if#1\nb \def\tmp{\nb#2}\fi}
884: \def\ignoretwo #1#2{}
```

\dotoc: 7, 4, 14, 39 \indexbuffer: 33, 37–39, 41–42 \doindex: 7, 4, 6, 14, 39–40, 44

The \doindexentry macro removes the \- characters from the control sequence \-⟨word⟩ by the \ignoretwo, so the \tmp includes ⟨word⟩ only. If the ⟨word⟩ begins by backslash, it is replaced by \nb using \remakebackslash macro. The reason: we needn't the backslash in the PDF internal labels, see 5.9 for more information. The index entry is finally printed by the \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 \reffile is read by \addbookmark ⟨node⟩ macro (see \reftocline and \refdg macros). The parameter ⟨node⟩ can be the number of section or the ⟨section⟩.⟨subsection⟩ pair. The ⟨node⟩ for which we are calculating children is saved in \currb macro. The \currsecb includes the parent of the \currb, if it exists. The \bk:⟨node⟩ is advanced by one using the \addbookmark macro.

docby.tex

```
\def\addbookmark#1{\undef{bk:#1}\iftrue\defsec{bk:#1}{1}%
   \else \tempnum=\csname bk:#1\endcsname\relax
         \advance\tempnum by1
         \edefsec{bk:#1}{\the\tempnum}
   \fi}
\def\currb{} % vychozi hodnota <uzel> pro jistotu
```

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

docby.tex

```
\def\bookmarks{\ifx\pdfoutput\undefined \else
   \bgroup
     \def\dotocline##1##2##3{%
         \undef{bk:##1}\iftrue \tempnum=0 \else \tempnum=\csname bk:##1\endcsname\relax\fi
         \if^^X##1^^X\advance\partnum by1
            \setoutline[sec:\thepart]{##2}{\opartname\space\thepart:  }%
         \else \setoutline[sec:##1]{##2}{}\fi}
     \def\ptocentry##1##2##3{\edef\tmpb{\ifx^^X##3^^X##2\else##3\fi}%
         \tempnum=0 \setoutline[##1##2]{\tmpb}{}}%
     \def\nb{\string\\}\def\TeX{TeX}\def\docbytex{DocBy.TeX}\def\_{_}\def\tt{}\def~{ }%
     \def\unskip{}\bookmarkshook
     \ifx\headtile\empty \else
        \tempnum=0 \setoutline[sec::start]{...\headtitle\empty...}{}\fi % viz \savepglink
     \tocbuffer
   \egroup \fi
}
```

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

docby.tex

```
\def\setoutline[#1]#2#3{{\def\nb{/_}\xdef\tmp{#1}}%
   \def\tmpa{\pdfoutline goto name{\tmp} count -\tempnum}%
   \cnvbookmark{\tmpa{#3\nobraces#2{\end}}}%
}
\def\cnvbookmark#1{#1}  % zadna konverze
\def\nobraces#1#{#1\nobrA}
\def\nobrA#1{\ifx\end#1\empty\else\nobraces#1\fi}
```

The special "conversion" macro \cnvbookmark is used here. It is nonactive by default. User can set (for example) \let\cnvbookmark=\lowercase for č → c, ž → z etc. conversions. The \lccode setting can be done by \bookmarkshook.

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

5.11 Sorting by Alphabetical Order

This work is done by \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

\ignoretwo: 31, 37, 39, 42 \remakebackslash: 39 \addbookmark: 38–40 \currb: 38–40
\currsecb: 38 \bookmarks: 7, 36, 40 \setoutline: 40 \cnvbookmark: 14, 40 \nobraces: 40
\nobrA: 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 \ifAleB 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 \fi are balanced by the \fif macro in nested \if...\fi constructions. This save the number of \expandafter commands.

docby.tex

```
\newif\ifAleB
\def\nullbuf{\def\indexbuffer{}}
\def\return#1#2\fi\relax{#1} \def\fif{\fi}
```

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.

docby.tex

```
\def\sortindex{\expandafter\nullbuf
  \expandafter\mergesort\indexbuffer\end,\end
}
```

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 bdeikmnz,. 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 936) and only \end is in the second group. The \gobblerest macro removes the second \end from input queue.

docby.tex

```
\def\mergesort #1#2,#3{%
   \ifx,#1                      % prazdna-skupina,neco,  (#2=neco #3=pokracovani)
      \addtext#2,\to\indexbuffer    % dvojice skupin vyresena
      \return{\fif\mergesort#3}%    % \mergesort pokracovani
   \fi
   \ifx,#3                      % neco,prazna-skupina,  (#1#2=neco #3=,)
      \addtext#1#2,\to\indexbuffer  % dvojice skupin vyresena
      \return{\fif\mergesort}%      % \mergesort dalsi
   \fi
   \ifx\end#3                   % neco,konec (#1#2=neco)
      \ifx\empty\indexbuffer                  % neco=kompletni setrideny seznam
         \edef\indexbuffer{\napercarky#1#2\end}%  % vlozim \, mezi polozky
         \return{\fif\fif\gobblerest}%            % koncim
      \else                     % neco=posledni skupina nebo \end
         \return{\fif\fif \expandafter\nullbuf    % spojim \indexbuffer+neco a cele znova
               \expandafter\mergesort\indexbuffer#1#2,#3}%
   \fi\fi                    % zatriduji:  p1+neco1,p2+neco2, (#1#2=p1+neco1 #3=p2)
   \isAleB #1#3\ifAleB       % p1<p2
      \addtext#1\to\indexbuffer   % p1 do bufferu
      \return{\fif\mergesort#2,#3}%        % \mergesort neco1,p2+neco2,
   \else                     % p1>p2
      \addtext#3\to\indexbuffer   % p2 do bufferu
      \return{\fif\mergesort#1#2,}%        % \mergesort p1+neco1,neco2,
   \fi
   \relax % zarazka, na ktere se zastavi \return
}
```

The core of the \mergesort is on the lines 943–948. 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 927–933: we need to save the rest of the nonempty group to the output \indexbuffer

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

and go to the next pair of groups. If the terminal string `\end,\end` is scanned then the next run of `\mergesort` is executed after `\indexbuffer` is put to the input queue and set it to empty value.

The sorting of the two items are realized by `\isAleB` ⟨*itemA*⟩⟨*itemB*⟩ macro. The items are in the form `\-`⟨*wordA*⟩ and `\-`⟨*wordB*⟩. The macro converts these parameters to the strings by `\string` primitive and expands to `\testAleB` ⟨*wordA*⟩`\relax`⟨*wordB*⟩`\relax`. The `\lowercase` primitive is executed here because we needn't distinguish between uppercase/lowercase letters.

docby.tex

```
952: \def\isAleB #1#2{%
953:    \edef\tmp{\expandafter\ignoretwo\string#1&0\relax\expandafter\ignoretwo\string#2&1\relax}%
954:    \lowercase \expandafter {\expandafter \testAleB \tmp}%
955: }
```

The `\testAleB` ⟨*wordA*⟩`\relax`⟨*wordB*⟩`\relax` macro tests if ⟨*wordA*⟩ precedes ⟨*wordB*⟩. It he 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 953.

docby.tex

```
956: \def\testAleB #1#2\relax #3#4\relax {%
957:    \ifx #1#3\testAleB #2\relax #4\relax \else
958:        \ifnum `#1<`#3 \AleBtrue \else \AleBfalse \fi
959:    \fi
960: }
```

The macro `\napercarky` inserts `\,` separators between items in the sorted `\indexbuffer`.

docby.tex

```
961: \def\napercarky#1{\ifx#1\end \else
962:     \noexpand\,\noexpand#1\expandafter\napercarky
963:   \fi
964: }
```

5.12 Merging of the List of the Page Numbers

Each occurrence of the ⟨*word*⟩ is stored to the `\reffile` as `\refuseword {`⟨*word*⟩`}{`⟨*page*⟩`}`. This macro is processed at the begin of the document when `\reffile` is read.

docby.tex

```
969: \def\refuseword#1#2{%
970:    \expandafter \ifx\csname w:#1\endcsname \relax
971:       \defsec{w:#1}{#2}
972:    \else
973:       \edefsec{w:#1}{\csname w:#1\endcsname,#2}
974:    \fi
975: }
```

So, the list of the pages where the ⟨*word*⟩ occurs is stored in the `\w:`⟨*word*⟩ macro. Pages are separated by commas. The list looks like:

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

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 ⟨*from*⟩`--`⟨*to*⟩. This work is done by `\listofpages {`⟨*word*⟩`}` macro which puts the list of pages to the input queue terminated by `,0,` and executes the `\transf` macro.

docby.tex

```
976: \def\listofpages#1{%
977:     \expandafter\expandafter\expandafter \transf\csname w:#1\endcsname,0,%
978: }
```

The `\transf` macro removes the page numbers which are equal to `\dgnum` or `\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:

docby.tex

```
980: \newcount\apinum
981: \newcount\dgnum
982: \newcount\tempnum
983: \newif\ifdash
```

\isAleB: 41–42 \testAleB: 42 \napercarky: 41–42 \refuseword: 25, 30, 37–38, 42
\listofpages: 20–21, 42 \dgnum: 20–21, 42–43 \apinum: 20–21, 42–43

```
\newif\iffirst
```

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.

docby.tex

```
\def\transf{\dashfalse \firsttrue \tempnum=-100 \bgroup \cykltransf}

\def\cykltransf #1,{\ifnum #1=\apinum \else \ifnum #1=\dgnum \else
   \ifnum #1=0 \let\cykltransf=\egroup
      \ifdash \pglink\the\tempnum\relax \fi
   \else
      \ifnum #1=\tempnum % cislo se opakuje, nedelam nic
      \else
         \advance\tempnum by 1
         \ifnum #1=\tempnum  % cislo navazuje
            \ifdash \else
               --\dashtrue
            \fi
         \else               % cislo nenavazuje
            \ifdash
               \advance\tempnum by-1
               \pglink\the\tempnum \relax\dashfalse, \pglink#1\relax
            \else
               \carka \pglink#1\relax
   \fi\fi\fi\fi
   \tempnum=#1 \fi\fi \cykltransf
}
\def\carka{\iffirst \firstfalse \else, \fi}
```

The \cykltransf macro is a little finite state automaton. It needs no more comments.

5.13 Multicolumn typesetting

The macros for multicolumn printing are borrowed from "TEXbook inside out", pages 244–246.

docby.tex

```
\newdimen\colsep  \colsep=\parindent % horiz.  mezera mezi sloupci
\def\roundtolines #1{%% zaokrouhlí na celé násobky vel.  rádku
   \divide #1 by\baselineskip \multiply #1 by\baselineskip}
\def\corrsize #1{%% #1 := #1 + \splittopskip - \topskip
   \advance #1 by \splittopskip \advance #1 by-\topskip}

\def\begmulti #1 {\par\bigskip\penalty0 \def\Ncols{#1}
   \splittopskip=\baselineskip
   \setbox0=\vbox\bgroup\penalty0
   \advance\hsize by\colsep
   \divide\hsize by\Ncols  \advance\hsize by-\colsep}
\def\endmulti{\vfil\egroup \setbox1=\vsplit0 to0pt
   \calculatedimone
   \ifdim \dimen1<2\baselineskip
     \vfil\break \dimen1=\vsize \corrsize{\dimen1} \fi
   \dimen0=\Ncols\baselineskip \advance\dimen0 by-\baselineskip
   \advance\dimen0 by \ht0 \divide\dimen0 by\Ncols
   \roundtolines{\dimen0}%
   \ifdim \dimen0>\dimen1 \splitpart
   \else \makecolumns{\dimen0} \fi
   \ifvoid0 \else \errmessage{ztracený text ve sloupcích?} \fi
   \bigskip}
\def\makecolumns#1{\setbox1=\hbox{}\tempnum=0
   \loop \ifnum\Ncols>\tempnum
      \setbox1=\hbox{\unhbox1 \vsplit0 to#1 \hss}
      \advance\tempnum by1
   \repeat
   \hbox{}\nobreak\vskip-\splittopskip \nointerlineskip
   \line{\unhbox1\unskip}}
\def\splitpart{\roundtolines{\dimen1}
```

\transf: 42–43 \cykltransf: 43

```
1042:    \makecolumns{\dimen1} \advance\dimen0 by-\dimen1
1043:    \vskip 0pt plus 1fil minus\baselineskip \break
1044:    \dimen1=\vsize \corrsize{\dimen1}
1045:    \ifvoid0 \else
1046:       \ifdim \dimen0>\dimen1 \splitpart
1047:       \else \makecolumns{\dimen0} \fi \fi}% TBN
```

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 863).

docby.tex

```
1048: \def\calculatedimone{%
1049:    \ifdim\pagegoal=\maxdimen \dimen1=\vsize \corrsize{\dimen1}
1050:    \else \dimen1=\pagegoal \advance\dimen1 by-\pagetotal \fi}
```

5.14 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.

docby.tex

```
1055: \catcode`\_=12
1056: \let\subori=\_ \def\_{_}
1057: \everymath={\catcode`\_=8 } \everydisplay={\catcode`\_=8 }
```

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

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

docby.tex

```
1063: \catcode`\"=\active
1064: \let\activeqq="
1065: \def"{\leavevmode\hbox\bgroup\mubytein=1\let\leftcomment=\empty
1066:       \let\returntoBlack=\empty \let\linecomment=\empty \let"=\egroup
1067:       \def\par{\errmessage{\string\par\space inside \string"...\string"}}%
1068:       \setverb\tt\quotehook
1069: }
```

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

docby.tex

```
1071: \def\langleactive{\uccode`\~=`\<\catcode`\<=13
1072:       \uppercase{\def~}##1>{{$\langle$\it##1\/$\rangle$}}}
```

6 Index

The control sequences marked by (≻) are sequences at user level. Other control sequences are internal in DocBy.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 pagenumber in the list of page numbers. This means the page where the sequence is documented at user level.