OpTeX

Format Based on Plain TeX and OPmac

Petr Olšák, 2020, 2021

http://petr.olsak.net/optex

OpTeX is LuaTeX format with Plain TeX and OPmac. Only LuaTeX engine is supported.

OpTeX should be a modern Plain TeX with power from OPmac (Fonts Selection System, colors, graphics, references, hyperlinks, indexing, bibliography, ...) with preferred Unicode fonts.

The main goal of OpTeX is:

- OpTeX keeps the simplicity (like in Plain TeX and OPmac macros).
- There is no old obscurities concerning various 8-bit encodings and various engines.
- OpTeX provides a powerful Fonts Selection System (for Unicode font families, of course).
- OpTeX supports hyphenations of all languages installed in your TeX system.
- All features from OPmac macros are copied. For example sorting words in the Index, reading .bib files directly, syntax highlighting, colors, graphics, hyperlinks, references).
- Macros are documented in the same place where code is.
- User namespace of control sequences is separated from the internal namespace of OpTeX and primitives (\foo versus \_foo). The namespaces for macro writers are designed too.

If you need to customize your document or you need to use something very specific, then you can copy relevant parts of OpTeX macros into your macro file and do changes to these macros here. This is a significant difference from LaTeX or ConTeXt, which is an attempt to create a new user level with a plenty of non-primitive parameters and syntax hiding TeX internals. The macros from OpTeX are simple and straightforward because they solve only what is explicitly needed, they do not create a new user level for controlling your document. We are using TeX directly in this case. You can use OpTeX macros, understand them, and modify them.

OpTeX offers a markup language for authors of texts (like LaTeX), i.e. the fixed set of tags to define the structure of the document. This markup is different from the LaTeX markup. It may offer to write the source text of the document somewhat clearer and more attractive.

The manual includes two parts: user documentation and technical documentation. The second part is generated directly from the sources of OpTeX. There are many hyperlinks from one part to second and vice versa.

This manual describes OpTeX features only. We suppose that the user knows TeX basics. They are described in many books. You can see a short document TeX in nutshell too.

---

1 OPmac package is a set of simple additional macros to Plain TeX. It enables users to take advantage of LaTeX functionality but keeps Plain TeX simplicity. See http://petr.olsak.net/opmac-e.html for more information about it.

2 All these features are implemented by TeX macros, no external program is needed.
Contents

1 User documentation

1.1 Starting with OpTEX ........................................... 5
1.2 Page layout ................................................. 5
  1.2.1 Setting the margins ...................................... 5
  1.2.2 Concept of the default page ............................. 6
  1.2.3 Footnotes and marginal notes ............................ 7
1.3 Fonts .......................................................... 7
  1.3.1 Font families ............................................. 7
  1.3.2 Font sizes ................................................ 8
  1.3.3 Typesetting math ....................................... 9
1.4 Typical elements of the document ............................ 10
  1.4.1 Chapters and sections ................................... 10
  1.4.2 Another numbered objects ............................... 10
  1.4.3 References ............................................... 12
  1.4.4 Hyperlinks, outlines .................................... 12
  1.4.5 Lists ..................................................... 13
  1.4.6 Tables ................................................... 14
  1.4.7 Verbatim ................................................ 16
1.5 Autogenerated lists .......................................... 18
  1.5.1 Table of contents ...................................... 18
  1.5.2 Making the index ....................................... 18
  1.5.3 Bib\TeX\xing ........................................... 20
1.6 Graphics ..................................................... 21
  1.6.1 Colors ................................................ 21
  1.6.2 Images ................................................ 22
  1.6.3 PDF transformations ..................................... 22
  1.6.4 Ovals, circles ......................................... 23
  1.6.5 Putting images and texts wherever .................... 24
1.7 Others ....................................................... 24
  1.7.1 Using more languages ................................... 24
  1.7.2 Pre-defined styles ..................................... 25
  1.7.3 Loading other macro packages .......................... 25
  1.7.4 Lorem ipsum dolor sit .................................. 26
  1.7.5 Logos .................................................. 26
  1.7.6 The last page .......................................... 26
  1.7.7 Use Op\TeX ............................................ 27
1.8 Summary ..................................................... 27
1.9 API for macro writers ....................................... 28
1.10 Compatibility with Plain \TeX ............................... 29

2 Technical documentation ........................................ 30

2.1 The main initialization file ................................ 30
2.2 Concept of namespaces of control sequences ............... 32
  2.2.1 Prefixing internal control sequences ................ 32
  2.2.2 Namespace of control sequences for users .......... 32
  2.2.3 Macro files syntax ................................... 33
  2.2.4 Name spaces for package writers .................... 33
  2.2.5 Summary about rules for external macro files published for Op\TeX .......................... 33
  2.2.6 The implementation of the namespaces ............... 34
2.20.3 Implementation ......................................................... 109
2.21 The .ref file ............................................................... 113
2.22 References ..................................................................... 115
2.23 Hyperlinks .................................................................... 116
2.24 Making table of contents .................................................. 118
2.25 PDF outlines ................................................................. 120
  2.25.1 Nesting PDF outlines ................................................ 120
  2.25.2 Strings in PDF outlines .............................................. 121
2.26 Chapters, sections, subsections ......................................... 123
2.27 Lists, items .................................................................... 128
2.28 Verbatim, listings ........................................................... 130
  2.28.1 Inline and “display” verbatim ..................................... 130
  2.28.2 Listings with syntax highlighting ............................... 134
2.29 Graphics ...................................................................... 138
2.30 The \table macro, tables and rules ................................... 143
  2.30.1 The boundary declarator : ........................................ 143
  2.30.2 Usage of the \tabskip primitive .................................. 144
  2.30.3 Tables to given width ............................................... 144
  2.30.4 \eqbox: boxes with equal width across the whole document ...... 145
  2.30.5 Implementation of the \table macro and friends ............. 145
2.31 Balanced multi-columns ................................................... 150
2.32 Citations, bibliography .................................................... 152
  2.32.1 Macros for citations and bibliography preloaded in the format ... 152
  2.32.2 The \usebib command ............................................... 155
  2.32.3 Notes for bib-style writers ........................................ 156
  2.32.4 The usebib.opm macro file loaded when \usebib is used .......... 157
  2.32.5 Usage of the bib-iso690 style .................................... 160
  2.32.6 Implementation of the bib-iso690 style ....................... 166
2.33 Sorting and making Index .................................................. 170
2.34 Footnotes and marginal notes .......................................... 176
2.35 Styles ....................................................................... 178
  2.35.1 \report and \letter styles ................................. 178
  2.35.2 \slides style for presentations ............................. 179
2.36 Logos ..................................................................... 183
2.37 Multilingual support ...................................................... 183
  2.37.1 Lowercase, uppercase codes .................................... 183
  2.37.2 Hyphenations ......................................................... 184
  2.37.3 Multilingual phrases and quotation marks ..................... 187
2.38 Other macros .............................................................. 190
2.39 Lua code embedded to the format ...................................... 192
  2.39.1 General ................................................................. 192
  2.39.2 Allocators ............................................................. 192
  2.39.3 Callbacks ............................................................. 193
  2.39.4 Handling of colors using attributes ......................... 198
2.40 Printing documentation ................................................... 201
  2.40.1 Implementation ....................................................... 202

Index 206
Chapter 1
User documentation

1.1 Starting with OpTeX

OpTeX is compiled as a format for LuaTeX. Maybe there is a command `optex` in your \TeX distribution. Then you can write into the command line

\texttt{optex document}

You can try to process `optex op-demo` or `optex optex-doc`.

If there is no `optex` command, see more information about installation OpTeX at \url{http://petr.olsak.net/optex}.

A minimal document should be:

\texttt{\fontfam[LMfonts] Hello World! \bye}

The first line \texttt{\fontfam[LMfonts]} tells that Unicode Latin Modern fonts (derived from Computer Modern) are used. If you omit this line then preloaded Latin Modern fonts are used but preloaded fonts cannot be in Unicode\footnote{This is a technical limitation of Lua\TeX for fonts downloaded in formats: only 8bit fonts can be preloaded.}. So the sentence \texttt{Hello World} will be OK without the first line, but you cannot print such sentence in other languages (for example \texttt{Ahoj světe!}) where Unicode fonts are needed because the characters like \texttt{ě} are not mapped correctly in preloaded fonts.

A somewhat larger example with common settings should be:

\texttt{\fontfam[Termes] % selecting Unicode font family Termes (section 1.3.1)\typosize[11/13] % setting default font size and baselineskip (sec. 1.3.2)\margins/1 a4 (1,1,1,1)in % setting A4 paper, 1 in margins (section 1.2.1)\cslang % Czech hyphenation patterns (section 1.7.1)\newpage Tady je zkušební textík v českém jazyce. \bye}

You can look at \texttt{op-demo.tex} file for a more complex, but still simple example.

1.2 Page layout

1.2.1 Setting the margins

The \texttt{\margins} command declares margins of the document. This command have the following parameters:

\texttt{\margins\langle pg\rangle \langle fmt\rangle (\langle left\rangle,\langle right\rangle,\langle top\rangle,\langle bot\rangle)\langle unit\rangle}

\texttt{\example:\margins/1 a4 (2.5,2.5,2,2)cm}

Parameters are:

\begin{itemize}
  \item \texttt{\langle pg\rangle} ... 1 or 2 specifies one-page or two-pages design.
  \item \texttt{\langle fmt\rangle} ... paper format (a4, a4l, a5, letter, etc. or user defined).
  \item \texttt{\langle left\rangle, \langle right\rangle, \langle top\rangle, \langle bot\rangle} ... gives the amount of left, right, top and bottom margins.
  \item \texttt{\langle unit\rangle} ... unit used for values \texttt{\langle left\rangle, \langle right\rangle, \langle top\rangle, \langle bot\rangle}.
\end{itemize}
Each of the parameters ⟨left⟩, ⟨right⟩, ⟨top⟩, ⟨bot⟩ can be empty. If both ⟨left⟩ and ⟨right⟩ are nonempty then \hsize is set. Else \hsize is unchanged. If both ⟨left⟩ and ⟨right⟩ are empty then typesetting area is centered in the paper format. The analogical rule works when ⟨top⟩ or ⟨bot⟩ parameter is empty (\vsize instead \hsize is used). Examples:

\margins/1 a4 (,,,)mm % \hsize, \vsize untouched, \% typesetting area centered
\margins/1 a4 (,2,,)cm % right margin set to 2cm \% \hsize, \vsize untouched, vertically centered

If ⟨pg⟩=1 then all pages have the same margins. If ⟨pg⟩=2 then the declared margins are true for odd pages. The margins at the even pages are automatically mirrored in such case, it means that ⟨left⟩ is replaced by ⟨right⟩ and vice versa.

OpTEX declares following paper formats: a4, a4l (landscape a4), a5, a5l, a3, a3l, b5, letter and user can declare another own format by \sdef:

\sdef{_pgs:b5l}{(250,176)mm}
\sdef{_pgs:letterl}{(11,8.5)in}

The ⟨fmt⟩ can be also in the form ⟨⟨width⟩,⟨height⟩⟩⟨unit⟩ where ⟨⟨unit⟩⟩ is optional. If it is missing then ⟨⟨unit⟩⟩ after margins specification is used. For example:

\margins/1 (100,200) (7,7,7,7)mm
declares the paper 100×200 mm with all four margins 7 mm. The spaces before and after ⟨fmt⟩ parameter are necessary.

The command \magscale[⟨factor⟩] scales the whole typesetting area. The fixed point of such scaling is the upper left corner of the paper sheet. Typesetting (breakpoints etc.) is unchanged. All units are relative after such scaling. Only paper format's dimensions stay unscaled. Example:

\margins/2 a5 (22,17,19,21)mm
\magscale[1414] \margins/1 a4 (,,,)mm

The first line sets the \hsize and \vsize and margins for final printing at a5 format. The setting on the second line centers the scaled typesetting area to the true a4 paper while breaking points for paragraphs and pages are unchanged. It may be usable for review printing. After the review is done, the second line can be commented out.

1.2.2 Concept of the default page

OpTEX uses “output routine” for page design. It is very similar to the Plain TEX output routine. There is \headline followed by “page body” followed by \footline. The \headline is empty by default and it can be used for running headers repeated on each page. The \footline prints centered page number by default. You can set the \footline to empty using \nopagenumbers macro.

The margins declared by \margins macro (documented in the previous section 1.2.1) is concerned to the page body, i.e. the \headline and \footline are placed to the top and bottom margins.

The distance between the \headline and the top of the page body is given by the \headlinedist register. The distance between bottom of the page body and the \footline is given by \footlinedist. The default values are:

\headline = {} \% \hss\_rmfixed \_folio \_hss \% \folio expands to page number
\footline = \{\_hss\_rmfixed \_folio \_hss\} \% \hss expands to page number
\headlinedist = 14pt \% from baseline of \headline to top of page body
\footlinedist = 24pt \% from last line in pagebody to baseline of footline
The page body should be divided into top insertions (floating tables and figures) followed by a real text and followed by footnotes. Typically, the only real text is here.

The \texttt{\pgbackground} tokens list is empty by default but it can be used for creating a background of each page (colors, picture, watermark for example). The macro \texttt{\draft} uses this register and puts big text DRAFT as a watermark to each page. You can try it.

More about the page layout is documented in sections 2.7.4 and 2.18.

1.2.3 Footnotes and marginal notes

The Plain \TeX{}'s macro \texttt{\footnote} can be used as usual. But a new macro \texttt{\fnote{\langle text \rangle}} is defined. The footnote mark is added automatically and it is numbered on each chapter from one\footnote{You can declare \texttt{\fnotenumglobal} if you want footnotes numbered in whole document from one or \texttt{\fnotenumpages} if you want footnotes numbered at each page from one. Default setting is \texttt{\fnotenumchapters}}. The \langle text \rangle is scaled to 80 %. User can redefine footnote mark or scaling, as shown in the section 2.34.

The \fnote macro is fully applicable only in “normal outer” paragraph. It doesn’t work inside boxes (tables, for example). If you are solving such a case then you can use the command \texttt{\fnotemark\langle numeric-label \rangle} inside the box: only the footnote mark is generated here. When the box is finished you can use \texttt{\fnotetext\langle text \rangle}. This macro puts the \langle text \rangle to the footnote. The \langle numeric-label \rangle has to be 1 if only one such command is in the box. Second \fnotemark inside the same box has to have the parameter 2 etc. The same number of \fnotetexts have to be written after the box as the number of \fnotemarks inserted inside the box. Example:

```
Text in a paragraph\fnote{First notice}... % a "normal" footnote
\table{...}{...\fnotemark1...\fnotemark2...} % two footnotes in a box
\fnotetext{Second notice}
\fnotetext{Third notice}
...
\table{...}{...\fnotemark1...} % one footnote in a box
\fnotetext{Fourth notice}
```

The marginal note can be printed by the \texttt{\mnote\langle text \rangle} macro. The \langle text \rangle is placed to the right margin on the odd pages and it is placed to the left margin on the even pages. This is done after second \TeX{} run because the relevant information is stored in an external file and read from it again. If you need to place the notes only to the fixed margin write \texttt{\fixmnotes\right} or \texttt{\fixmnotes\left}.

The \langle text \rangle is formatted as a little paragraph with the maximal width \texttt{\mnotesize} ragged left on the left margins or ragged right on the right margins. The first line of this little paragraph has its vertical position given by the position of \texttt{\mnote} in the text. The exceptions are possible by using the \texttt{up} keyword: \texttt{\mnote up\langle dimen \rangle\langle text \rangle}. You can set such \langle dimen \rangle to each \texttt{\mnote} manually in final printing in order to margin notes do not overlap. The positive value of \langle dimen \rangle shifts the note up and negative value shifts it down. For example \texttt{\mnote up 2\baselineskip\langle text \rangle} shifts this marginal note two lines up.

1.3 Fonts

1.3.1 Font families

You can select the font family by \texttt{\fontfam\langle Family-name \rangle}. The argument \langle Family-name \rangle is case insensitive and spaces are ignored in it. For example, \texttt{\fontfam[LM Fonts]} is equal to \texttt{\fontfam[LMfonts]} and it is equal to \texttt{\fontfam[lmfonts]}. Several aliases are prepared, thus \texttt{\fontfam[Latin Modern]} can be used for loading Latin Modern family too.

If you write \texttt{\fontfam[?]} then all font families registered in Op\TeX{} are listed on the terminal and in the log file. If you write \texttt{\fontfam[catalog]} then a catalog of all fonts registered in
OpTeX and available in your \TeX{} system is printed. The instructions on how to register your own font family are appended in the catalog.

If the family is loaded then *font modifiers* applicable in such font family are listed on the terminal: \texttt{(\texttt{\caps{}}, \texttt{\cond{}} for example). And there are four basic *variant selectors* (\texttt{\rm{}, \bf{}, \it{}, \bi{}}). The usage of variant selectors is the same as in Plain \TeX{}: {\it{italics text}}, {\bf{bold text}} etc.

The font modifiers (\texttt{\caps{}, \cond{}} for example) can be used before a variant selector and they can be (independently) combined: \texttt{\caps{}}it or \texttt{\cond{}}\caps{\bf{}}. The modifiers keep their internal setting until the group ends or until another modifier that negates the previous feature is used. So \texttt{\caps{}} \texttt{\rm{ First text \it{ Second text}}} gives \texttt{FIRST TEXT SECOND TEXT}.

The font modifier without following variant selector does not change the font actually, it only prepares data used by next variant selectors. There is one special variant selector \texttt{\currvar{}} which does not change the selected variant but reloads the font due to (maybe newly specified) font modifier(s).

The context between variants \texttt{\rm{}} ↔ \texttt{\it{}} and \texttt{\bf{}} ↔ \texttt{\bi{}} is kept by the \texttt{\em} macro (emphasis text). It switches from current \texttt{\rm{}} to \texttt{\it{}}, from current \texttt{\it{}} to \texttt{\rm{}}, from current \texttt{\bf{}} to \texttt{\bi{}} and from current \texttt{\bi{}} to \texttt{\bf{}}. The italics correction \texttt{\sl} is inserted automatically, if needed.

Example:

\begin{verbatim}
\it{This is \em{important} text.} % = This is \texttt{\it{ important \sl}}
\bf{This is \em{important} text.} % = This is \texttt{\bf{ important \sl}}
\end{verbatim}

More about the \TeX{} Font Selection System is written in the technical documentation in the section \texttt{2.13}. You can mix more font families in your document, you can declare your own variant selectors or modifiers, etc.

### 1.3.2 Font sizes

The command \texttt{\typosize{⟨fontsize⟩/⟨baselineskip⟩}} sets the font size of text and math fonts and baselineskip. If one of these two parameters is empty, the corresponding feature stays unchanged. Don’t write the unit of these parameters. The unit is internally set to \texttt{\ptunit} which is 1pt by default. You can change the unit by the command \texttt{\ptunit=⟨something-else⟩}, for instance \texttt{\ptunit=1mm} enlarges all font sizes declared by \texttt{\typosize}. Examples:

\begin{verbatim}
\typosize[10/12] % default of Plain \TeX
\typosize[11/12.5] % font 11pt, baseline 12.5pt
\typosize[8/] % font 8pt, baseline unchanged
\end{verbatim}

The commands for font size setting described in this section have local validity. If you put them into a group, the settings are lost when the group is finished. If you set something relevant with paragraph shape (baselineskip given by \texttt{\typosize} for example) then you must first finalize the paragraph before closing the group: {\texttt{\typosize[12/14] \texttt{\ldots{text of paragraph}\ldots{\par}.

The command \texttt{\typoscale{⟨font-factor⟩/⟨baselineskip-factor⟩}} sets the text and math fonts size and baselineskip as a multiple of the current fonts size and baselineskip. The factor is written in “scaled”-like way, it means that 1000 means factor one. The empty parameter is equal to the parameter 1000, i.e. the value stays unchanged. Examples:

\begin{verbatim}
\typoscale[800/800] % fonts and baselineskip re-size to 80 %
\typoscale[magstep2/1] % fonts bigger 1,44times (\magstep2 expands to 1440)
\end{verbatim}

First usage of \texttt{\typosize} or \texttt{\typoscale} macro in your document sets so-called *main values*, i.e. main font size and main baselineskip. They are internally saved in registers \texttt{\mainfosize} and \texttt{\mainbaselineskip}. 8
The \texttt{\textbackslash typoscale} command does scaling with respect to current values by default. If you want to do it with respect to the main values, type \texttt{\textbackslash scalemain} immediately before \texttt{\textbackslash typoscale} command.

\texttt{\textbackslash typosize[12/14.4]} \% first usage in document, sets main values internally
\texttt{\textbackslash typosize[15/18]} \% bigger font
\texttt{\textbackslash scalemain \textbackslash typoscale[800/800]} \% reduces from main values, no from current.

The \texttt{\textbackslash typosize} and \texttt{\textbackslash typoscale} macros initialize the font family by \texttt{\textbackslash rm}. You can re-size only the current font by the command \texttt{\textbackslash thefontsize\{\langle font-size\rangle\}} or the font can be rescaled by \texttt{\textbackslash thefontsize\{\langle factor\rangle\}}. These macros don’t change math fonts sizes nor baselineskip.

There is “low level” \texttt{\textbackslash setfontsize\{\langle size-spec\rangle\}} command which behaves like a font modifier and sets given font size used by next variant selectors. It doesn’t change the font size immediately, but the following variant selector does it. For example \texttt{\textbackslash setfontsize\{at15pt\}\textbackslash currvar} sets current variant to 15pt.

If you are using a font family with “optical sizes feature” (i.e. there are more recommended sizes of the same font which are not scaled linearly; a good example is Computer Modern aka Latin Modern fonts) then the recommended size is selected by all mentioned commands automatically.

More information about resizing of fonts is documented in the section 2.12.

### 1.3.3 Typesetting math

See the additional document \texttt{Typesetting Math with Op\TeX} for more details about this issue.

Op\TeX\ preloads a collection of 7bit Computer Modern math fonts and AMS fonts in its format for math typesetting. You can use them in any size and in the \texttt{\textbackslash boldmath} variant. Most declared text font families (see \texttt{\textbackslash fontfam} in the section 1.3.1) are configured with a recommended Unicode math font. This font is automatically loaded unless you specify \texttt{\textbackslash noloadmath} before first \texttt{\textbackslash fontfam} command. See log file for more information about loading text font family and Unicode math fonts. If you prefer another Unicode math font, specify it by \texttt{\textbackslash loadmath\{\langle font-file\rangle\}} or \texttt{\textbackslash loadmath\{\langle font-name\rangle\}} before first \texttt{\textbackslash fontfam} command.

Hundreds math symbols and operators like in AMSTeX are accessible. For example \texttt{\alpha}, \texttt{\geq}, \texttt{\sum \sum}, \texttt{\sphericalangle \sphericalangle}, \texttt{\bumpeq \bumpeq}. See AMSTeX manual or \texttt{Typesetting Math with Op\TeX} for complete list of math symbols.

The following math alphabets are available:

\texttt{\mit} \% mathematical variables \texttt{abc–xyz, ABC–XYZ}
\texttt{\it} \% text italics \texttt{abc–xyz, ABC–XYZ}
\texttt{\rm} \% text roman \texttt{abc–xyz, ABC–XYZ}
\texttt{\cal} \% normal caligraphics \texttt{ABC–XYZ}
\texttt{\script} \% script \texttt{ABCDEFGHIJKLMNOPQRSTUVWXYZ}
\texttt{\frak} \% fracture \texttt{abc–xyz, ABC–XYZ}
\texttt{\bbchar} \% double stroked letters \texttt{ABCDEFGHIJKLMNOPQRSTUVWXYZ}
\texttt{\bf} \% sans serif bold \texttt{abc–xyz, ABC–XYZ}
\texttt{\bi} \% sans serif bold slanted \texttt{abc–xyz, ABC–XYZ}

The last two selectors \texttt{\bf} and \texttt{\bi} select the sans serif fonts in math regardless of the current text font family. This is a common notation for vectors and matrices. You can re-declare them, see section 2.16.2 where definitions of Unicode math variants of \texttt{\bf} and \texttt{\bi} selectors are documented.

The math fonts can be scaled by \texttt{\textbackslash typosize} and \texttt{\textbackslash typoscale} macros. Two math fonts collections are prepared: \texttt{\normalmath} for normal weight and \texttt{\boldmath} for bold. The first one is set by default, the second one is usable for math formulae in titles typeset in bold, for example.
You can use \text{⟨text⟩} inside math mode. It behaves as {\hbox{⟨text⟩}} (i.e. the ⟨text⟩ is printed in horizontal non-math mode) but the size of the ⟨text⟩ is adapted to the context of math size (text or script or scriptscript).

1.4 Typical elements of the document

1.4.1 Chapters and sections

The documents can be divided into chapters (\chap), sections (\sec), subsections (\secc) and they can be titled by \tit command. The parameters are separated by the end of current line (no braces are used):

\tit Document title \ (end of line)
\chap Chapter title \ (end of line)
\sec Section title \ (end of line)
\secc Subsection title \ (end of line)

The chapters are automatically numbered by one number, sections by two numbers (chapter.section), and subsections by three numbers. If there are no chapters then sections have only one number and subsections two.

The implicit design of the titles of chapter etc. is implemented in the macros \_printchap, \_printsec and \_printsecc. A designer can simply change these macros if he/she needs another behavior.

The first paragraph after the title of chapter, section, and subsection is not indented but you can type \let\_firstnoindent=\relax if you need all paragraphs indented.

If a title is so long then it breaks into more lines in the output. It is better to hint at the breakpoints because \TeX does not interpret the meaning of the title. Users can put the \nl (means newline) to the breakpoints.

If you want to arrange a title to more lines in your source file then you can use \^\^J at the end of each line (except the last one). When \^\^J is used, then the reading of the title continues at the next line. The “normal” comment character % doesn’t work in titles. You can use \nl\^\^J if you want to have corresponding lines in the source and the output.

The chapter, section, or subsection isn’t numbered if the \nonum precedes. And the chapter, section, or subsection isn’t delivered to the table of contents if \notoc precedes. You can combine both prefixes.

1.4.2 Another numbered objects

Apart from chapters, sections, and subsections, there are another automatically numbered objects: equations, captions for tables and figures. The user can declare more numbered objects.

If the user writes the \eqmark as the last element of the display mode then this equation is numbered. The equation number is printed in brackets. This number is reset in each section by default.

If the \eqalignno is used, then user can put \eqmark to the last column before \cr. For example:

\eqalignno{
a^2+b^2 &= c^2 \cr
c &\approx \sqrt{a^2+b^2} \& \eqmark \cr}

Another automatically numbered object is a caption which is tagged by \caption/t for tables and \caption/f for figures. The caption text follows. The \cskip can be used between \caption text and the real object (table or figure). You can use two orders: ⟨caption⟩\cskip ⟨object⟩ or ⟨object⟩\cskip ⟨caption⟩. The \cskip creates appropriate vertical space between them. Example:
The dependency of the computer-dependency on the age.

\begin{table}[!h]
\centering
\begin{tabular}{rl}
\hline
age & value \\
0--1 & unmeasured \\
1--6 & observable \\
6--12 & significant \\
12--20 & extremal \\
20--40 & normal \\
40--60 & various \\
60--\infty & moderate \\
\hline
\end{tabular}
\caption{The dependency of the computer-dependency on the age.}
\end{table}

This example produces:

\textbf{Table 1.4.1} The dependency of the computer-dependency on the age.

\begin{verbatim}
<table>
<thead>
<tr>
<th>age</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0--1</td>
<td>unmeasured</td>
</tr>
<tr>
<td>1--6</td>
<td>observable</td>
</tr>
<tr>
<td>6--12</td>
<td>significant</td>
</tr>
<tr>
<td>12--20</td>
<td>extremal</td>
</tr>
<tr>
<td>20--40</td>
<td>normal</td>
</tr>
<tr>
<td>40--60</td>
<td>various</td>
</tr>
<tr>
<td>60--\infty</td>
<td>moderate</td>
</tr>
</tbody>
</table>
\end{verbatim}

You can see that the word “Table” followed by a number is added by the macro \caption{t}. The caption text is centered. If it occupies more lines then the last line is centered.

The macro \caption{f} behaves like \caption{t} but it is intended for figure captions with independent numbering. The word (Table, Figure) depends on the selected language (see section 1.7.1 about languages).

If you wish to make the table or figure as a floating object, you need to use Plain \TeX macros \midinsert or \topinsert terminated by \endinsert. Example:

\begin{verbatim}
\topinsert % table and its caption printed at the top of the current page
<caption and table>
\endinsert
\end{verbatim}

The pair \midinsert...\endinsert prefers to put the enclosed object to the current place. Only if this is unable due to page breaking, it behaves like \topinsert...\endinsert.

There are five prepared counters A, B, C, D and E. They are reset in each chapter and section\footnote{This feature can be changed, see the section 2.26 in the technical documentation.}. They can be used in context of \numberedpar \langle letter \rangle \langle text \rangle macro. For example:

\begin{verbatim}
\def\theorem {\numberedpar A{Theorem}}
\def\corollary {\numberedpar A{Corollary}}
\def\definition {\numberedpar B{Definition}}
\def\example {\numberedpar C{Example}}
\end{verbatim}

Three independent numbers are used in this example. One for Theorems and Corollaries second for Definitions and third for Examples. The user can write \theorem \textit{Let $\mathcal{M}$ be...} and the new paragraph is started with the text: \textbf{Theorem 1.4.1}. Let $M$ be... You can add an optional parameter in brackets. For example, \theorem [[(L'Hôpital's rule)] Let $f$, $g$ be... is printed like \textbf{Theorem 1.4.2 (L'Hôpital's rule)}. Let $f$, $g$ be...
1.4.3 References

Each automatically numbered object documented in sections 1.4.1 and 1.4.2 can be referenced if optional parameter \([\text{label}]\) is appended to \texttt{\chap, \sec, \secc, \caption/t, \caption/f} or \texttt{\eqmark}. The alternative syntax is to use \texttt{\label[\text{label}]} before mentioned commands (not necessarily directly before). The reference is realized by \texttt{\ref[\text{label}]} (prints the number of the referenced object) or \texttt{\pgref[\text{label}]} (prints the page number). Example:

\begin{verbatim}
\sec[beatle] About Beatles
\noindent\hfil\table{rl}{...} % the table
cskip
\caption/t [comp-depend] The dependency of the comp-dependency on the age.
\label[pythagoras]
$$ a^2 + b^2 = c^2 \eqmark $$
\end{verbatim}

Now we can point to the section-\ref[beatle] on the page-\pgref[beatle] or write something about the equation-\ref[pythagoras]. Finally there is an interesting Table-\ref[comp-depend].

The text printed by \texttt{\ref} or \texttt{\pgref} can be given explicitly by \texttt{\ref[\text{label}]\{\text{text}\}} or \texttt{\pgref[\text{label}]\{\text{text}\}}. If the \texttt{\text{text}} includes the @ character, it is replaced by implicitly printed text. Example: see \texttt{\ref[lab]\{section-@\}} prints the same as see section-\ref[lab], but first case creates larger active area for mouse clicking, when \texttt{\hyperlinks} are declared.

If there are forward referenced objects then users have to run \TeX{} twice. During each pass, the working *.ref file (with references data) is created and this file is used (if it exists) at the beginning of the document.

You can use the \texttt{\label[\text{label}]} before the \texttt{\theorem}, \texttt{\definition} etc. (macros defined with \texttt{\numberedpar}) if you want to reference these numbered objects. You can’t use \texttt{\theorem[\text{label}]} because the optional parameter is reserved to another purpose here.

You can create a reference to whatever else by commands \texttt{\label[\text{label}]}\{\text{text}\}. The connection between \texttt{\text{label}} and \texttt{\text{text}} is established. The \texttt{\ref[\text{label}]} will print \texttt{\text{text}}.

By default, labels are not printed, of course. But if you are preparing a draft version of your document then you can declare \texttt{\showlabels}. The labels are printed at their destination places after such a declaration.

1.4.4 Hyperlinks, outlines

If the command \texttt{\hyperlinks (color-in) (color-out)} is used at the beginning of the document, then the following objects are hyperlinked in the PDF output:

- numbers and texts generated by \texttt{\ref} or \texttt{\pgref},
- numbers of chapters, sections, subsections, and page numbers in the table of contents,
- numbers or marks generated by \texttt{\cite} command (bibliography references),
- texts printed by \texttt{\url} or \texttt{\ulink} commands.

The last object is an external link and it is colored by \texttt{(color-out)}. Other links are internal and they are colored by \texttt{(color-in)}. Example:

\hyperlinks \Blue \Green % internal links blue, URLs green.

You can use another marking of active links: by frames which are visible in the PDF viewer but invisible when the document is printed. The way to do it is to define the macros \texttt{\_pgborder, \_tocborder, \_citeborder, \_refborder} and \texttt{\_urlborder} as the triple of RGB components of the used color. Example:
\def\_tocborder {1 0 0} % links in table of contents: red frame
\def\_pgborder {0 1 0} % links to pages: green frame
\def\_citeborder {0 0 1} % links to references: blue frame

By default, these macros are not defined. It means that no frames are created.

The hyperlinked footnotes can be activated by \fnotelinks (color-fnt) (color-fnf) where footnote marks in the text have (color-fnt) and the same footnote marks in footnotes have (color-fnf). You can define relevant borders \_fntborder and \_fnfborder analogically as \_pgborder (for example).

There are “low level” commands to create the links. You can specify the destination of the internal link by \dest[type]:⟨label⟩\{⟨text⟩\}. The active text linked to the \dest can be created by \ilink[type]:⟨label⟩\{⟨text⟩\}. The ⟨type⟩ parameter is one of the toc, pg, cite, ref, or another special for your purpose. These commands create internal links only when \hyperlinks is declared.

The \url macro prints its parameter in \tt font and creates a potential breakpoints in it (after slash or dot, for example). If the \hyperlinks declaration is used then the parameter of \url is treated as an external URL link. An example: \url{http://www.olsak.net} creates http://www.olsak.net. The characters %, \, #, { and } must be escaped in the ⟨url⟩ parameter.

The PDF format provides outlines which are notes placed in the special frame of the PDF viewer. These notes can be managed as a structured and hyperlinked table of contents of the document. The command \outlines{⟨level⟩} creates such outlines from data used for the table of contents in the document. The ⟨level⟩ parameter gives the level of opened sub-outlines in the default view. The deeper levels can be opened by mouse click on the triangle symbol after that.

If you are using a special unprotected macro in section titles then \outlines macro may crash. You must declare a variant of the macro for outlines case which is expandable. Use \regmacro in this case. See the section 1.5.1 for more information about \regmacro.

The command \insertoutline{⟨text⟩} inserts a next entry into PDF outlines at the main level 0. These entries can be placed before the table of contents (created by \outlines) or after it. Their hyperlink destination is in the place where the \insertoutline macro is used.

The command \thisoutline{⟨text⟩} uses ⟨text⟩ in the outline instead of default title text for the first following \chap, \sec, or \secc. Special case: \thisoutline\relax doesn’t create any outline for the following \chap, \sec, or \secc.

1.4.5 Lists

The list of items is surrounded by \begitems and \enditems commands. The asterisk (*) is active within this environment and it starts one item. The item style can be chosen by the \style parameter written after \begitems:

\style o % small bullet
\style 0 % big bullet (default)
\style - % hyphen char
\style n % numbered items 1., 2., 3., ...
\style N % numbered items 1), 2), 3), ...
\style i % numbered items (i), (ii), (iii), ...
\style I % numbered items I, II, III, IV, ...
\style a % items of type a), b), c), ...

---

4 More exactly, there are the same rules as for \code command, see section 1.4.7.
For example:
\begitems
* First idea
* Second idea in subitems:
  \begitems \style i
  * First sub-idea
  * Second sub-idea
  * Last sub-idea
\enditems
* Finito
\enditems

produces:

- First idea
- Second idea in subitems:
  (i) First sub-idea
  (ii) Second sub-idea
  (iii) Last sub-idea
- Finito

Another style can be defined by the command \sdef{\_item:\{style\}}{\{text\}}. Default item can be set by \defaultitem{\{text\}}. The list environments can be nested. Each new level of items is indented by next multiple of \indent value which is set to \parindent by default. The \ilevel register says what level of items is currently processed. Each \begitems starts \everylist tokens register. You can set, for example:

\everylist={\ifcase\ilevel\or \style X \or \style x \else \style - \fi}

You can say \begitems \novspaces if you don’t want vertical spaces above and below the list. The nested item list is without vertical spaces automatically. More information about the design of lists of items should be found in the section 2.27.

A “selected block of text” can be surrounded by \begblock...\endblock. The default design of blocks of text is indented text in smaller font. The blocks of text can be nested.

1.4.6 Tables

The macro \table{\{declaration\}}{\{data\}} provides similar \{declaration\} of tables as in \TEX: you can use letters l, r, c, each letter declares one column (aligned to left, right, center, respectively). These letters can be combined by the | character (vertical line). Example
\table{|l|c|r||}
  \crl
  Month & commodity & price \crli \tskip2pt
  January & notebook & \$ 700 \ cr
  February & skateboard & \$ 100 \ cr
  July & yacht & k\$ 170 \crl

generates the result:

<table>
<thead>
<tr>
<th>Month</th>
<th>commodity</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>notebook</td>
<td>$ 700</td>
</tr>
<tr>
<td>February</td>
<td>skateboard</td>
<td>$ 100</td>
</tr>
<tr>
<td>July</td>
<td>yacht</td>
<td>k$ 170</td>
</tr>
</tbody>
</table>
Apart from l, r, c declarators, you can use the \p{⟨size⟩} declarator which declares the column with paragraphs of given width. More precisely, a long text in the table cell is printed as a multiline paragraph with given width. By default, the paragraph is left-right justified. But there are alternatives:

- \p{⟨size⟩}\fl fit left, i.e. left justified, ragged right,
- \p{⟨size⟩}\fr fit right, i.e. right justified, ragged left,
- \p{⟨size⟩}\fc fit center, i.e. ragged left plus right,
- \p{⟨size⟩}\fs fit special, short one-line paragraph centered, long paragraph normal,
- \p{⟨size⟩}\fx fit extra, left-right justified but last line centered.

You can use (⟨text⟩) in the ⟨declaration⟩. Then this text is applied in each line of the table. For example r(\kern10pt)l adds more 10pt space between r and l rows.

An arbitrary part of the ⟨declaration⟩ can be repeated by a ⟨number⟩ prefixed. For example 3c means ccc or c 3{c} means c|c|c|c. Note that spaces in the ⟨declaration⟩ are ignored and you can use them in order to more legibility.

The command \cr used in the ⟨data⟩ part of the table is generally known from Plain \TeX. It marks the end of each row in the table. Moreover Op\TeX defines following similar commands:

- \cr... the end of the row with a horizontal line after it.
- \crll... the end of the row with a double horizontal line after it.
- \crli... like \cr but the horizontal line doesn’t intersect the vertical double lines.
- \crlli... like \crli but horizontal line is doubled.
- \crlp{⟨list⟩}... like \crli but the lines are drawn only in the columns mentioned in comma-separated ⟨list⟩ of their numbers. The ⟨list⟩ can include ⟨from⟩–⟨to⟩ declarators, for example \crlp{1-3,5} is equal to \crlp{1,2,3,5}.

The \tskip(⟨dimen⟩) command works like the \noalign{\vskip(⟨dimen⟩)} immediately after \cr* commands but it doesn’t interrupt the vertical lines.

You can use the following parameters for the \table macro. Default values are listed too.

\everytable={} % code used in \vbox before table processing
\thistable={} % code used in \vbox, it is removed after using it
\tabiteml={\enspace} % left material in each column
\tabitemr={\enspace} % right material in each column
\tabstrut={\strut} % strut which declares lines distance in the table
\tablinespace=2pt % additional vert. space before/after horizontal lines
\vkv kern=ipt % space between lines in double vertical line
\hhk kern=ipt % space between lines in double horizontal line
\tabskip=0pt % space between columns
\tabskipl=0pt \tabskipr=0pt % space before first and after last column

Example: if you do \tabiteml={\enspace}\tabitemr={\enspace} then the \table acts like \LaTeX’s array environment.

If there is an item that spans to more than one column in the table then the macro \multispan{⟨number⟩} (from Plain \TeX) can help you. Another alternative is the command \mspan(⟨number⟩){⟨declaration⟩}{⟨text⟩} which spans ⟨number⟩ columns and formats the ⟨text⟩ by the ⟨declaration⟩. The ⟨declaration⟩ must include a declaration of only one column with the same syntax as common \table ⟨declaration⟩. If your table includes vertical rules and you want to create continuous vertical rules by \mspan, then use rule declarators \ | after c, l or r letter in \mspan ⟨declaration⟩. The exception is only in the case when \mspan includes the first column and the table have rules on the left side. The example of \mspan usage is below.

The \frame{⟨text⟩} makes a frame around ⟨text⟩. You can put the whole \table into \frame if you need double-ruled border of the table. Example:
creates the following result:

<table>
<thead>
<tr>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
</tr>
<tr>
<td>seven</td>
</tr>
</tbody>
</table>

The \vspan\{number\}\{\text{\text} \} shifts the \{\text{\text} \} down in order it looks like to be in the center of the \{\text{\text} \} lines (current line is first). You can use this for creating tables like in the following example:

\thisstable{\vrule height 20pt depth10pt width0pt} \vspan\{\vrule height 20pt depth10pt width0pt\} \vspan\{\vrule height 20pt depth10pt width0pt\}

You can use \vspan with non-integer parameter too if you feel that the result looks better, for example \vspan2.1\{\text{\text} \}.

The rule width of tables and implicit width of all \vrule\s and \hrule\s can be set by the command \rulewidth=\{\text{\text} \}. The default value given by \TeX is 0.4 pt.

The : columns boundary declarator is described in section 2.30.1. The tables with given width can be declared by to\{\text{\text} \} or pxto\{\text{\text} \}. More about it is in section 2.30.3. Many tips about tables can be seen on the site http://petr.olsak.net/optex/optex-tricks.html.

1.4.7 Verbatim

The display verbatim text have to be surrounded by the \begtt and \endtt couple. The in-line verbatim have to be tagged (before and after) by a character which is declared by \verbchar\{\text{\text} \}. For example \verbchar`\{\text{\text} \} declares the character ` for in-line verbatim markup. And you can use \`\relax\{\text{\text} \} for verbatim \relax (for example). Another alternative of printing in-line verbatim text is \code\{\text{\text} \} (see below).

If the numerical register \ttline is set to the non-negative value then display verbatim will number the lines. The first line has the number \ttline+1 and when the verbatim ends then the \ttline value is equal to the number of the last line printed. Next \begtt\ldots\endtt environment will follow the line numbering. Op\TeX\ sets \ttline=-1 by default.
The indentation of each line in display verbatim is controlled by \ttindent register. This register is set to the \parindent by default. Users can change the values of the \parindent and \ttindent independently.

The \begtt command starts the internal group in which the catcodes are changed. Then the \everytt tokens register is run. It is empty by default and the user can control fine behavior by it. For example, the catcodes can be re-declared here. If you need to define an active character in the \everytt, use \adef as in the following example:

\everytt={\adef!{?}\adef?{!}}

\begtt
Each occurrence of the exclamation mark will be changed to the question mark and vice versa. Really? You can try it!
\endtt

The \adef command sets its parameter as active after the parameter of \everytt is read. So you don’t have to worry about active categories in this parameter.

There is an alternative to \everytt named \everyintt which is used for in-line verbatim surrounded by an \verbchar or processed by the \code command.

The \everytt is applied to all \begtt...\endtt environments (if it is not declared in a group). There are tips for such global \everytt definitions here:

\everytt={\typsize[9/11]} % setting font size for verbatim
\everytt={\ttline=0} % each listing will be numbered from one
\everytt={\visiblesp} % visualization of spaces

If you want to apply a special code only for one \begtt...\endtt environment then don’t set any \everytt but put desired material at the same line where \begtt is. For example:

\begtt \adef!{?}\adef?{!}
Each occurrence of ? will be changed to ! and vice versa.
\endtt

The in-line verbatim surrounded by a \verbchar doesn’t work in parameter of macros and macro definitions. (It works in titles declared by \chap, \sec etc. and in \fnote, because these macros are specially defined in OpTEX). You can use more robust command \code{⟨text⟩} in problematic situations, but you have to escape the following characters in the ⟨text⟩: \, #, %, braces (if the braces are unmatched in the ⟨text⟩), and space or ^ (if there are more than one subsequent spaces or ^ in the ⟨text⟩). Examples:

\code{\text, \%\#} ... prints \text, %#
\code{a@{}*\&$} ... prints @{}*\&$ without escaping, but you can escape these characters too, if you want.
\code{a \ b} ... two spaces between a b, the second must be escaped
\code{xy\{z} ... xy{z ... unbalanced brace must be escaped
\code{\^M} ... prints \^M, the second ^ must be escaped

You can print verbatim listing from external files by the \verbininput command. Examples:

\verbininput (12-42) program.c % listing from program.c, only lines 12-42
\verbininput (-60) program.c % print from begin to the line 60
\verbininput (61-) program.c % from line 61 to the end
\verbininput (-) program.c % whole file is printed
\verbininput (70+10) program.c % from line 70, only 10 lines printed
\verbininput (+10) program.c % from the last line read, print 10 lines
\verbininput (-5+7) program.c % from the last line read, skip 5, print 7
\verbininput (+) program.c % from the last line read to the end
You can insert additional commands for \verbinput before the first opening bracket. They are processed in the local group. For example, \verbinput \hspace=20cm (-) program.c.

The \ttline influences the line numbering by the same way as in \begtt...\endtt environment. If \ttline=-1 then real line numbers are printed (this is the default). If \ttline<=-1 then no line numbers are printed.

The \verbinput can be controlled by \everytt, \ttindent just like in \begtt...\endtt.

The \begtt...\endtt pair or \verbinput can be used for listings of codes. Automatic syntax highlighting is possible, for example \begtt this\syntax{C} activates colors for C programs. Or \verbinput \syntax{HTML} (-) file.html can be used for HTML or XML codes. OpTEX implements C, Python, \TeX, HTML and XML syntax highlighting. More languages can be declared, see the section 2.28.2.

If the code is read by \verbinput and there are comment lines prefixed by two characters then you can set them by \commentchars\langle first\rangle\langle second\rangle. Such comments are fully interpreted by \TeX (i.e. not verbatim). Section 2.28.1 (page 133) says more about this feature.

1.5 Autogenerated lists

1.5.1 Table of contents

The \maketoc command prints the table of contents of all \chap, \sec and \secc used in the document. These data are read from the external \*.ref file, so you have to run \TeX more than once (typically three times if the table of contents is at the beginning of the document).

Typically, we don’t want to repeat the name of the section “Table of contents” in the table of contents again. The direct usage of \chap or \sec isn’t recommended here because the table of contents is typically not referenced to itself. You can print the unnumbered and unreferenced title of the section like this:

\nonum\notoc\sec Table of Contents

If you need a customization of the design of the TOC, read the section 2.24.

If you are using a special macro in section or chapter titles and you need different behavior of such macro in other cases then use \regmacro{\langle case-toc\rangle}{\langle case-mark\rangle}{\langle case-outline\rangle}. The parameters are applied locally in given cases. The \regmacro can be used repeatedly: then its parameters are accumulated (for more macros). If a parameter is empty then original definition is used in given case. For example:

\% default value of \mylogo macro used in text and in the titles:
\def\mylogo{\leavevmode\hbox{{\Red\it My}{\setfontsize{mag1.5}\rm Lo}Go}}
\% another variants:
\regmacro {\def\mylogo{\hbox{\Red My\Black LoGo}}}{\% used in TOC
\{\def\mylogo{\hbox{{\it My}\/LoGo}}} {\% used in running heads
\{\def\mylogo{MyLoGo}}} {\% used in PDF outlines

1.5.2 Making the index

The index can be included in the document by the \makeindex macro. No external program is needed, the alphabetical sorting is done inside \TeX at macro level.

The \ii command (insert to index) declares the word separated by the space as the index item. This declaration is represented as an invisible item on the page connected to the next visible word. The page number of the page where this item occurs is listed in the index entry. So you can type:

The \ii resistor resistor is a passive electrical component ...

You cannot double the word if you use the \iid instead of \ii:
The \iid resistor is a passive electrical component ...

or:

Now we'll deal with the \iid resistor.

Note that the dot or comma has to be separated by space when \iid is used. This space (before dot or comma) is removed by the macro in the current text.

The multiple-words entries are commonly arranged in the index as follows:

linear dependency 11, 40–50
— independency 12, 42–53
— space 57, 76
— subspace 58

To do this you have to declare the parts of the index entries by the / separator. Example:

\bf Definition.
\ii linear/space,vector/space
\em Linear space} (or \em vector space}) is a nonempty set of...

The number of the parts of one index entry (separated by /) is unlimited. Note, that you can spare your typing by the comma in the \ii parameter. The previous example is equivalent to \ii linear/space \ii vector/space.

Maybe you need to propagate to the index the similar entry to the linear/space in the form of space/linear. You can do this by the shorthand ,@ at the end of the \ii parameter. Example:

\ii linear/space,vector/space,@
is equivalent to:
\ii linear/space,vector/space \ii space/linear,space/vector

If you really need to insert the space into the index entry, write ~.

The \ii or \iid commands can be preceded by \iitype ⟨letter⟩, then such reference (or more references generated by one \ii) has the specified type. The page numbers of such references should be formatted specially in the index. OpTeX implements only \iitype b, \iitype i and \iitype u: the page number in bold or in italics or underlined is printed in the index when these types are used. The default index type is empty, which prints page numbers in normal font. The \TeXbook index is a good example.

The \makeindex creates the list of alphabetically sorted index entries without the title of the section and without creating more columns. Op\TeXX provides other macros \begmulti and \endmulti for more columns:

\begmulti ⟨number of columns⟩
⟨text⟩
\endmulti

The columns will be balanced. The Index can be printed by the following code:

\sec Index
\begmulti 3 \makeindex \endmulti

Only “pure words” can be propagated to the index by the \ii command. It means that there cannot be any macro, \TeX primitive, math selector, etc. But there is another possibility to create such a complex index entry. Use “pure equivalent” in the \ii parameter and map this equivalent to a real word that is printed in the index. Such mapping is done by \iis command. Example:

The \ii chiquadrat $\chi$-quadrat method is ...
If the \ii relax `\relax` command is used then \TeX/ is relaxing.

...
\textit{\chi$\chi$-quadrat} \\relax

The \textit{\chi$\chi$-quadrat} \\relax \{\textit{text}\} creates one entry in the “dictionary of the exceptions”. The sorting is done by the \textit{\chi$\chi$-quadrat} but the \textit{text} is printed in the index entry list.

The sorting rules when \texttt{makeindex} runs depends on the current language. See section 1.7.1 about languages selection.

### 1.5.3 Bib\TeX\textit{Xing}

The command \texttt{\cite\{\textit{label}\}} (or \texttt{\cite\{\textit{label-1},\textit{label-2},\ldots,\textit{label-n}\}}) creates the citation in the form \cite{42} (or [15, 19, 26]). If \texttt{\shortcitations} is declared at the beginning of the document then continuous sequences of numbers are re-printed like this: [3–5, 7, 9–11]. If \texttt{\sortcitations} is declared then numbers generated by one \texttt{\cite} command are sorted upward.

If \texttt{\nonumcitations} is declared then the marks instead of numbers are generated depending on the used bib-style. For example, the citations look like [Now08] or [Nowak, 2008].

The \texttt{\rcite\{\textit{labels}\}} creates the same list as \texttt{\cite\{\textit{labels}\}} but without the outer brackets. Example: \cite{tbn}, pg.~13 creates [4, pg. 13].

The \texttt{\ecite\{\textit{label}\}\{\textit{text}\}} prints the \textit{text} only, but the entry labeled \textit{label} is decided as to be cited. If \texttt{\hyperlinks} is used then \textit{text} is linked to the references list.

You can define alternative formatting of \texttt{\cite} command. Example:

\begin{verbatim}
\def\cite[#1]{(\rcite[#1])} % \cite\{\textit{label}\} creates (27)
\def\cite[#1]{$^{\rcite[#1]}$} % \cite\{\textit{label}\} creates^{27}
\end{verbatim}

The numbers printed by \texttt{\cite} correspond to the same numbers generated in the list of references. There are two possibilities to generate this references list:

- Manually using \texttt{\bib\{\textit{label}\}} commands.
- By \texttt{\usebib/\{\textit{type}\} \{\textit{style}\} \{\textit{bib-base}\}} command which reads \*.bib files directly.

Note that another two possibilities documented in OPmac (using external Bib\TeX\textit{X} program) isn’t supported because Bib\TeX\textit{X} is an old program that does not support Unicode. And Biber seems to be not compliant with Plain \TeX\textit{X}.

#### References created manually using \texttt{\bib\{\textit{label}\}} command.

\begin{verbatim}
\bib tst P. Olšák. \textit{Typografický systém \TeX{}}
\end{verbatim}

If you are using \texttt{\nonumcitations} then you need to declare the \textit{\marks} used by \texttt{\cite} command. To do it you must use long form of the \texttt{\bib} command in the format \texttt{\bib[\textit{\label}] = \{\textit{\marks}\}}. The spaces around equal sign are mandatory. Example:

\begin{verbatim}
\bib tbn = {Olšák, 2001}
\end{verbatim}

#### Direct reading of \*.bib files is possible by \texttt{\usebib} macro.

This macro reads and uses macro package \texttt{librarian.tex} by Paul Isambert. The usage is:

\begin{verbatim}
\usebib/c \{\textit{\style}\} \{\textit{\bib-base}\} \% sorted by \texttt{\cite-order} (c=cite),
\usebib/s \{\textit{\style}\} \{\textit{\bib-base}\} \% sorted by style (s=style).
% example:
\nocite[*] \usebib/s (simple) op-biblist \% prints all from op-biblist.bib
\end{verbatim}

The \textit{\bib-base} is one or more \*.bib database source files (separated by spaces and without extension) and the \textit{\style} is the part of the filename \texttt{bib-\{\textit{\style}\}.opm} where the formatting of
the references list is defined. OpTeX supports simple or iso690 styles. The features of the iso690 style is documented in the section 2.32.5 in detail. The \usebib command is more documented in section 2.32.2.

Not all records are printed from ⟨bib-base⟩ files: the command \usebib selects only such bib-records which were used in \cite or \nocite commands in your document. The \nocite behaves as \cite but prints nothing. It tells only that the mentioned bib-record should be printed in the reference list. If \nocite[*] is used then all records from ⟨bib-base⟩ are printed.

You can create more independent lists of references (you are creating proceedings, for example). Use \bibpart \{⟨name⟩\} to set the scope where \cite s and references list are printed (and interconnected) independent of another parts of your document. The \cite labels used in different parts can be the same and they are not affected. References lists can be created manually by \bib or from a database by \usebib. Example:

\bibpart {AA}
\cite[⟨labelX⟩] ... \cite[⟨labelY⟩] ... % They belong to AA bib-list
\usebib/c (simple) file.bib % generates AA bib-list numbered 1, 2, ...
% \cite prints [1], [2], ... by bib-list AA

\bibpart {BB}
\cite[⟨labelZ⟩] ... \cite[⟨labelX⟩] ... % They belong to BB bib-list
\bibnum=0 \usebib/c (simple) my.bib % generates BB bib-list numbered 1, 2, ...
% \cite prints [1], [2], ... by bib-list BB

By default, \bibpart is empty. So \cite s and the references list are conneted using this empty internal name.

1.6 Graphics

1.6.1 Colors

OpTeX provides a small number of color selectors: \Blue, \Red, \Brown, \Green, \Yellow, \Cyan, \Magenta, \White, \Grey, \LightGrey and \Black. More such selectors can be defined by setting four CMYK components (using \setcmykcolor), or three RGB components (using \setrgbcolor) or one grey component (using \setgreycolor). For example

\def \Orange {\setcmykcolor{0 0.5 1 0}}
\def \Purple {\setrgbcolor{1 0 1}}
\def \DarkGrey {\setgreycolor{.1}}

The command \morecolors reads more definitions of color selectors from the \hTeX file x11nam.def. There are about 300 color names like \DeepPink, \Chocolate etc. If there are numbered variants of the same name, then the letters B, C, etc. are appended to the name in OpTeX. For example \Chocolate is Chocolate1, \ChocolateB is Chocolate2 etc.

The color selectors work locally in groups by default. See the technical documentation, section 2.20 for more information.

The basic colors \Blue, \Red, \Cyan, \Yellow etc. are defined with CMYK components using \setcmykcolor. On the other hand, you can define a color with three RGB components and \morecolors defines such RGB colors. By default, the color model isn’t converted but only stored to PDF output for each used color. Thus, there may be a mix of color models in the PDF output which is not a good idea. You can overcome this problem by declaration \onlyrgb or \onlycmyk. Then only the selected color model is used for PDF output and if a used color is declared by another color model then it is converted. The \onlyrgb creates colors more bright (usable for computer presentations). On the other hand, CMYK makes colors more true\textsuperscript{5} for printing.

\textsuperscript{5} Printed output is more equal to the monitor preview especially if you are using ICC profile for your printer.
You can define your color by a linear combination of previously defined colors using \colordef. For example:

\colordef\myCyan{.3\Green+.5\Blue} \% 30 \% green, 50 \% blue, 20\% white
\colordef\DarkBlue{\Blue+.4\Black} \% Blue mixed with 40 \% of black
\colordef\myGreen{\Cyan+\Yellow} \% exact the same as \Green
\colordef\MyColor{.3\Orange+.5\Green+.2\Yellow}

The linear combination is done in CMYK subtractive color space by default (RGB colors used in \colordef argument are converted first). If the resulting component is greater than 1 then it is truncated to 1. If a convex linear combination (as in the last example above) is used then it emulates color behavior on a painter’s palette. You can use \rgbcolordef instead of \colordef if you want to mix colors in the additive RGB color space. If \onlyrgb is set then \colordef works like \rgbcolordef.

The following example defines the macro for colored text on colored background. Usage:
\coloron{background}{foreground}\{text\}

The \coloron macro can be defined as follows:
\def\coloron#1#2#3{\
\setbox0=\hbox{#2#3}\
\leavevmode\rlap{#1\strut\vrule width\wd0}\box0
}
\coloron\Yellow\Brown{Brown text on yellow background}

1.6.2 Images

The \inspic{filename}.\{extension\} or \inspic{filename}.\{extension\}\{space\} inserts the picture stored in the graphics file with the name \{filename\}.\{extension\} to the document. You can set the picture width by \picw=\{dimen\} before \inspic command which declares the width of the picture. The image files can be in the PNG, JPG, JBIG2 or PDF format.

The \picwidth is an equivalent register to \picw. Moreover, there is an \picheight register which denotes the height of the picture. If both registers are set then the picture will be (probably) deformed.

The image files are searched in \picdir. This token list is empty by default, this means that the image files are searched in the current directory. Example: \picdir=\{img/\} supposes that image files are in \img subdirectory. Note: the directory name must end by / in the \picdir declaration.

Inkscape\footnote{A powerful and free Wysiwyg editor for creating vector graphics.} is able to save a picture to PDF and labels of the picture to another file\footnote{Chose “Omit text in PDF and create LaTeX file” option.}. This second file should be read by \TeXt to print labels in the same font as document font. Op\TeX supports this feature by \linkinspic{filename}.pdf command. It reads and displays both: PDF image and labels generated by Inkscape.

If you want to create vector graphics (diagrams, schema, geometry skicing) then you can do it by Wysiwyg graphics editor (Inkscape, Geogebra for example), export the result to PDF and include it by \inspic. If you want to “program” such pictures then Tikz package is recommended. It works in Plain \TeX and Op\TeX.

1.6.3 PDF transformations

All typesetting elements are transformed by linear transformation given by the current transformation matrix. The \pdfsetmatrix{\{a\} \{b\} \{c\} \{d\}} command makes the internal multiplication with the current matrix so linear transformations can be composed. One linear transformation given by the \pdfsetmatrix above transforms the vector \{0,1\} to \{\{a\},\{b\}\}
and $[1,0]$ to $\langle c, d \rangle$. The stack-oriented commands \pdfsave and \pdfrestore gives a possibility of storing and restoring the current transformation matrix and the position of the current point. This position has to be the same from \TeX's point of view as from the transformation point of view when \pdfrestore is processed. Due to this fact the \pdfsave\rlap{\langle transformed text \rangle}\pdfrestore or something similar is recommended.

\TeX provides two special transformation macros \pdfscale and \pdfrotate:

\begin{verbatim}
\pdfscale\{(horizontal-factor)\}\{(vertical-factor)\}
\pdfrotate\{(angle-in-degrees)\}
\end{verbatim}

These macros simply call the properly \pdfsetmatrix command.

It is known that the composition of transformations is not commutative. It means that the order is important. You have to read the transformation matrices from right to left. Example:

First: \pdfsave \pdfrotate{30}\pdfscale{-2}{2}\rlap{text1}\pdfrestore
% text1 is scaled two times and it is reflected about vertical axis
% and next it is rotated by 30 degrees left.
second: \pdfsave \pdfscale{-2}{2}\pdfrotate{30}\rlap{text2}\pdfrestore
% text2 is rotated by 30 degrees left then it is scaled two times
% and reflected about vertical axis.
third: \pdfsave \pdfrotate{-15.3}\pdfsetmatrix{2 0 1.5 2}\rlap{text3}%%
\pdfrestore % first slanted, then rotated by 15.3 degrees right

This gives the following result. First: text1 second: text2 third: text3

You can see that \TeX knows nothing about dimensions of transformed material, it treats it as with a zero dimension object. The \transformbox\{(transformation)\}\{(text)\} macro solves the problem. This macro puts the transformed material into a box with relevant dimensions. The (transformation) parameter includes one or more transformation commands \pdfsetmatrix, \pdfscale, \pdfrotate with their parameters. The (text) is transformed text.

Example: \frame{\transformbox{\pdfscale{1}{1.5}\pdfrotate{-10}}{moj}} creates \texttt{moj}.

The \rotbox\{(deg)\}\{(text)\} is shortcut for \transformbox{\pdfrotate\{(deg)\}\{(text)\}}.

1.6.4 Ovals, circles

The \inoval\{(text)\} creates a box like this: text. Multiline text can be put in an oval by the command \inoval\vbox\{(text)\}. Local settings can be set by \inoval\{\settings\}\{(text)\} or you can re-declare global settings by \ovalparams=\{(settings)\}. The default settings are:

\begin{verbatim}
\ovalparams=\{\roundness=2pt \fcolor=Yellow \lcolor=Red \lwidth=0.5bp \shadow=N \overlapmargins=N \hhkern=0pt \vvkern=0pt\}
\end{verbatim}

The total distance from text to oval boundary is \hhkern+\roundness at the left and right sides and \vvkern+\roundness at the top and bottom sides of the text.

If you need to set a parameters for the (text) (color, size, font etc.), put such setting right in front of the (text): \inoval\{\text settings\}\{(text)\}. 23
The $\texttt{\textcircled{ratio=1.8}}\{\text{text}\}$ creates a box like this \texttt{text}. The \texttt{\textcircled{ratio}} parameter means width/height. The usage is analogical like for oval. The default parameters are

\begin{verbatim}
\circleparams={\ratio=1 \fcolor=Yellow \lcolor=Red \width=0.5bp \shadow=N \ignoremargins=N \hhkern=2pt \vvkern=2pt}
\end{verbatim}

The macros $\texttt{\textcircled{clipinoval}}(x)\langle y\rangle\{\text{text}\}$ and $\texttt{\textcircled{clipincircle}}$ (with the same parameters) print the \texttt{text} when a clipping path (oval or circle with given \texttt{\textcircled{with}} and \texttt{\textcircled{height}} shifted its center by \texttt{x} to right and by \texttt{y} to up) is used. The \texttt{\textcircled{roundness=5mm}} is default for $\texttt{\textcircled{clipincircle}}$ and user can change it. Example:

$\texttt{\textcircled{clipincircle} 3cm 3.5cm 6cm 7cm \{\picw=6cm \inspic{myphoto.jpg}}$.

1.6.5 Putting images and texts wherever

The $\texttt{\puttext}(x)\langle y\rangle\{\text{text}\}$ puts the \texttt{text} shifted by \texttt{x} right and by \texttt{y} up from the current point of typesetting and does not change the position of the current point. Assume a coordinate system with origin in the current point. Then $\texttt{\puttext}(x)\langle y\rangle\{\text{text}\}$ puts the text at the coordinates \texttt{(x}, \texttt{y)}. More exactly the left edge of its baseline is at that position.

The $\texttt{\putpic}(x)\langle y\rangle\{\text{image-file}\}$ (including extension) of \texttt{image-file} puts an image given by \texttt{\textcircled{image-file}} at given position (\texttt{x} to right and \texttt{y} up) when a clipping path (oval or circle with given \texttt{width} and \texttt{height}) at given position (its left-bottom corner). You can write \texttt{\nospec} instead with \texttt{\textcircled{width}} or \texttt{\textcircled{height}} if this parameter is not specified.

1.7 Others

1.7.1 Using more languages

\textsc{OpTeX} prepares hyphenation patterns for all languages if such patterns are available in your \textsc{TeX} system. Only USEnglish patterns (original from Plain \textsc{TeX}) are preloaded. Hyphenation patterns of all other languages are loaded on demand when you first use the $\texttt{\langlist}$ command in your document. For example $\texttt{\delang}$ for German, $\texttt{\pllang}$ for Polish. The \texttt{\langlist} is a shortcut of the language (mostly from ISO 639-1). You can list all available languages by $\texttt{\langlist}$ macro. This macro prints now:

en(UEnglish) enus(UEnglishmax) engh(UKenglish) it(italian) ia(Interlingua) id(Indonesian) es(Czech) sk(Slovak) de(nGerman) fr(French) pl(Polish) cy(Welsh) da(Danish) es(Spanish) sl(Slovenian) fi(Finnish) hu(Hungarian) tr(Turkish) et(Estonian) eu(Basque) ga(Irish) nb(Bokmal) no(Nynorsk) nl(Dutch) pt(Portuguese) ro(Romanian) hr(Croatian) sv(Swedish) it(Italian) ia(Interlingua) id(Indonesian) cs(Czech) sk(Slovak)

For compatibility with e-plain macros, there is the command $\texttt{\uselanguage\{\text{language}\}}$. The parameter \texttt{\textcircled{language}} is long-form of language name, i.e. $\texttt{\uselanguage\{Czech\}}$ works the same as $\texttt{\pllang}$. The \texttt{\uselanguage} parameter is case insensitive.

For compatibility with e-plain, there are macros $\texttt{\ehyph}$, $\texttt{\chyph}$, $\texttt{\shyph}$ which are equivalent to $\texttt{\enlang}$, $\texttt{\cslang}$ and $\texttt{\sklang}$.

You can switch between language patterns by $\texttt{\langlist}$ commands mentioned above. Default is $\texttt{\enlang}$. \textsc{OpTeX} generates three phrases used for captions and titles in technical articles or books: \"Chapter\", \"Table\" and \"Figure\". These phrases need to be known in used language and it depends on the previously used language selectors $\texttt{\langlist}$. \textsc{OpTeX} declares these words only for few languages: Czech, German, Spanish, French, Greek, Italian, Polish, Russian, Slovak and English. If you need to use these words in other languages or you want to auto-generate
more words in your macros, then you can declare it by \sdef or \_langw commands as shown in section 2.37.3.

The \makeindex command needs to know the sorting rules used in your language. Op\TeX defines only a few language rules for sorting: Czech, Slovak and English. How to declare sorting rules for more languages are described in the section 2.33.

If you declare \ iso-code\ quotes, then the control sequences \" and \' should be used like this: \"(quoted text)\" or \'(quoted text)\' (note that the terminating character is the same but it isn’t escaped). This prints language-dependent normal or alternative quotes around (quoted text). The language is specified by \ iso-code. Op\TeX declares quotes only for Czech, German, Spanish, French, Greek, Italian, Polish, Russian, Slovak and English (\csquotes, \dequotes, \enquotes, ...). You can simply define your own quotes as shown in section 2.37.3. The \" is used for quotes visually more similar to the " character which can be primary quotes or secondary quotes depending on the language rules. Maybe you want to alternate the meaning of these two types of quotes. Use \ iso-code\quotes\altquotes in such case.

1.7.2 Pre-defined styles

Op\TeX defines three style-declaration macros \report, \letter and \slides. You can use them at the beginning of your document if you are preparing these types of documents and you don’t need to create your own macros.

The \report declaration is intended to create reports. It sets default font size to 11 pt and \parindent (paragraph indentation) to 1.2 em. The \tit macro uses smaller font because we assume that “chapter level” will be not used in reports. The first page has no page number, but the next pages are numbered (from number 2). Footnotes are numbered from one in the whole document. The macro \author {authors} can be used when \report is declared. It prints \author in italics at the center of the line. You can separate authors by \nl to more lines.

The \letter declaration is intended to create letters. See the files op-letter-*.tex for examples. The \letter style sets default font size to 11 pt and \parindent to 0 pt. It sets half-line space between paragraphs. The page numbers are not printed. The \subject macro can be used, it prints the word “Subject:” or “Věc” (or something else depending on current language) in bold. Moreover, the \address macro can be used when \letter is declared. The usage of the \address macro looks like:

\begin{verbatim}
\address
  (first line of address)
  (second line of address)
  (etc.)
  (empty line)
\end{verbatim}

It means that you need not use any special mark at the end of lines: the ends of lines in the source file are the same as in printed output. The \address macro creates \vtop with address lines. The width of such \vtop is equal to the widest line used in it. So, you can use \hfill\address... to put the address box to the right side of the document. Or you can use \prefixed \address... to put \prefixed before the first line of the address.

The \slides style creates a simple presentation slides. See an example in the file op-slides.tex. Run optex op-slides.tex and see the documentation of \slides style in the file op-slides.pdf.

Analogical declaration macro \book is not prepared. Each book needs individual typographical care. You need to create specific macros for design.

1.7.3 Loading other macro packages

You can load more macro packages by \input{(file-name)} or by \load[(file-names)]. The first case (\input) is \TeX primitive command, it can be used in the alternative old syntax
\input (filename) too. The second case (\load) allows specifying a comma-separated list of included files. Moreover, it loads each macro file only once, it sets temporarily standard category codes during loading and it tries to load (filename).opm or (filename).tex or (filename), the first occurrence wins. Example:

\load [qrCode, scanbase]

does \texttt{\input qrCode.opm} and and \texttt{\input scanbase.tex}. It saves local information about the fact that these file names (qrCode, scanbase) were loaded, i.e. next \load will skip them.

It is strongly recommended to use the \load macro for loading external macros if you need them. On the other hand, if your source document is structured to more files (with individual chapters or sections), use simply the \input primitive.

The macro packages intended to OpTeX have the name *.opm. The following packages are distributed as part of OpTeX:

- \texttt{qrCode.opm} enables to create QR codes.
- \texttt{tikz.opm} does \texttt{\input tikz.tex}, i.e. loads TikZ. It adds OpTeX-specific code.
- \texttt{mte.opm} includes settings for microtypographic extensions (protrusions+expanding fonts).
- \texttt{vlna.opm} enables to protect of one-letter prepositions and more things automatically.
- \texttt{emoji.opm} defines \texttt{\emoji{name}} command for colored emoticons.
- \texttt{plain-at.opm} defines the old names from plain TeX.
- \texttt{pdfextra.opm} allows the use of many extra features from PDF standard (by M. Vlasák).

See these files in \texttt{optex/pkg/} or \texttt{optex/\pkgname} for more information about them. The packages may have their documentation, try \texttt{texdoc \pkgname}.

### 1.7.4 Lorem ipsum dolor sit

A designer needs to concentrate on the design of the output and maybe he/she needs material for testing macros. There is the possibility to generate a neutral text for such experiments. Use \texttt{\lorem[number]} or \texttt{\lorem[from]-[to]}. It prints a paragraph (or paragraphs) with neutral text. The numbers [number] or [from], [to] must be in the range 1 to 150 because there are 150 paragraphs with neutral text prepared for you. The \lipsum macro is equivalent to \texttt{\lorem}.

Example: \texttt{\lipsum[1-150]} prints all prepared paragraphs.

### 1.7.5 Logos

The control sequences for typical logos can be terminated by optional / which is ignored when printing. This makes logos more legible in the source file:

\begin{quote}
We are using \TeX/ because it is cool. \OpTeX/ is better than \LaTeX.
\end{quote}

### 1.7.6 The last page

The number of the last page (it may be different from the number of pages) is expanded by \texttt{\lastpage} macro. It expands to ? in first \TeX run and to the last page in next \TeX runs.

There is an example for footlines in the format “current page / last page”:

\begin{verbatim}
\footline={\hss \fixedrm \folio/\lastpage \hss}
\end{verbatim}

The \texttt{\lastpage} expands to the last \texttt{\folio} which is a decimal number or Roman numeral (when \texttt{\pageno} is negative). If you need to know the total pages used in the document, use \texttt{\totalpages} macro. It expands to zero (in first \TeX run) or to the number of all pages in the document (in next \TeX runs).
1.7.7 Use \texttt{OpTeX}

The command \texttt{\useOpTeX} (or \texttt{\useoptex}) does nothing in OpTeX but it causes an error (undefined control sequence) when another format is used. You can put it as the first command in your document:

\texttt{\useOpTeX \% we are using OpTeX format, no LaTeX :)}

1.8 Summary

\texttt{\tit Title (terminated by end of line)}
\texttt{\chap Chapter Title (terminated by end of line)}
\texttt{\sec Section Title (terminated by end of line)}
\texttt{\secc Subsection Title (terminated by end of line)}

\texttt{\maketoc \% table of contents generation}
\texttt{\ii item1,item2 \% insertion the items to the index}
\texttt{\makeindex \% the index is generated}
\texttt{\label [labname] \% link target location}
\texttt{\ref [labname] \% link to the chapter, section, subsection, equation}
\texttt{\pgref [labname] \% link to the page of the chapter, section, ...}

\texttt{\caption/t \% a numbered table caption}
\texttt{\caption/f \% a numbered caption for the picture}
\texttt{\eqmark \% a numbered equation}
\texttt{\begitems \% start a list of the items}
\texttt{\enditems \% end of list of the items}
\texttt{\begblock \% start a block of text}
\texttt{\endblock \% end of block of text}
\texttt{\begtt \% start a verbatim text}
\texttt{\endtt \% end verbatim text}
\texttt{\verbchar X \% initialization character X for in-text verbatim}
\texttt{\code \% another alternative for in-text verbatim}
\texttt{\verbinput \% verbatim extract from the external file}
\texttt{\begmulti num \% start multicolumn text (num columns)}
\texttt{\endmulti \% end multicolumn text}
\texttt{\cite [labnames] \% refers to the item in the lits of references}
\texttt{\rcite [labnames] \% similar to \cite but [] are not printed.}
\texttt{\sortcitations \% similar to \cite but [] are not printed.}
\texttt{\bib [labname] \% an item in the list of references}
\texttt{\load [filenames] \% loading macro files}
\texttt{\fontfam [FamilyName] \% selection of font family}
\texttt{\typosize [font-size/baselineskip] \% size setting of typesetting}
\texttt{\typoscale [factor-font/factor-baselineskip] \% size scaling}
\texttt{\thefontsize [size] \% current font size}
\texttt{\inspic file.ext \% insert a picture, extensions: jpg, png, pdf}
\texttt{\table {rule}{data} \% macro for the tables like in LaTeX}
\texttt{\fnote {text} \% footnote (local numbering on each page)}
\texttt{\amnote {text} \% note in the margin (left or right by page number)}
\texttt{\hyperlinks {color-in}{color-out} \% PDF links activate as clickable}
\texttt{\outlines {level} \% PDF will have a table of contents in the left tab}
\texttt{\magscale [factor] \% resize typesetting, line/page breaking unchanged}
\texttt{\margins/pg format (left, right, top, bottom)unit \% margins setting}
\texttt{\report \letter \slides \% style declaration macros}
1.9 API for macro writers

All TeX primitives and almost all OpTeX macros are accessible by two names: \texttt{\_foo} (public or user name space) and \texttt{\_foo} (private name space). For example \texttt{\_hbox} and \texttt{\_hbox} means the same TeX primitive. More about it is documented in section 2.2.

If this manual refers \texttt{\_foo} then \texttt{\_foo} equivalent exists too. For example, we mention the \texttt{\addto} macro below. The \texttt{\addto} equivalent exists too, but it is not explicitly mentioned here. If we refer only \texttt{\_foo} then its public equivalent does not exist. For example, we mention the \texttt{\_codedecl} macro below, so this macro is not available as \texttt{\codedecl}.

If you are writing a document or macros specific for the document, then use simply user namespace \texttt{\_foo}. If you are writing more general macros, then use private namespace \texttt{\_foo}, but you should declare your own namespace by \texttt{\_namespace} macro and you have to follow the naming discipline described in section 2.2.4.

The alphabetically sorted list of macros typically usable for macro writers follows. More information about such macros can be found in the technical documentation. You can use hyperlinks here in order to go to the appropriate place of the technical documentation.

\begin{itemize}
\item \texttt{\addto \macro{(text)}} adds \texttt{text} at the end of \texttt{macro} body.
\item \texttt{\def (char)\{\body\}} defines \texttt{char} active character with meaning \texttt{body}.
\item \texttt{\afterfi \{\text\}}\texttt{\_ifi} expands to \texttt{\_ifi} \texttt{text}.
\item \texttt{\bp \{\dimen \expression\}} expands TeX dimension to decimal number in \texttt{bp} without unit.
\item \texttt{\_codedecl \{sequence\} \{\info\}} is used at beginning of macro files.
\item \texttt{\colordef \macro \{\text \}} declares \texttt{macro} as color switch.
\item \texttt{\cs \{\string\}} expands \texttt{\string}.
\item \texttt{\_doc ... \_cod} encloses documentation text in the macro code.
\item \texttt{\eodef \macro \#1\{\body\}} defines \texttt{macro} with parameter separated to end of line.
\item \texttt{\_endcode} closes the part of macro code in macro files.
\item \texttt{\_endnamespace} closes name space declared by \texttt{\_namespace}.
\item \texttt{\_hbox \{\label\}\{\text\}} creates \texttt{\_hbox\{\text\}} with common width across whole document.
\item \texttt{\expr \{\text\}} expands to result of the \texttt{\text} with decimal numbers.
\item \texttt{\_isfont \text \{\text \}} \texttt{\_ifi} as font switch.
\item \texttt{\foreach \fa\text \{\sizespec\}} declares \texttt{\fa} as the same font switch like \texttt{\fa} at given \texttt{\sizespec}.
\item \texttt{\foreach \list \do \{\parameters\}\{\what\}} is expandable loop over \texttt{\list}.
\item \texttt{\_isfont \macro \{\parameters\}\{\what\}} declares expandable \texttt{\macro} as loop over \texttt{\list}.
\item \texttt{\_ sincer \{\from\}...\{\to\}\do \{\what\}} is expandable loop with numeric variable.
\item \texttt{\_incr \{\counter\}} increases and \texttt{\_decr \{\counter\}} decreases \texttt{\counter} by one globally.
\item \texttt{\_ignoreit \{\one\}} \texttt{\_ignoresecond \{\one\}\{\two\}} ignores given parameter.
\item \texttt{\_expandafter} \texttt{\_ignoreopt} \texttt{\_thadimen} expands to decimal number \texttt{\dimen} without \texttt{\pt}.
\item \texttt{\_isempty}, \texttt{\_istokempty}, \texttt{\_isequal}, \texttt{\_ismacro}, \texttt{\_isdefined}, \texttt{\_islist} \texttt{\_isfile}, \texttt{\_isfont} do various tests. Example: \texttt{\_islist\_list\{\text\}\iftrue} does \texttt{\iftrue} if \texttt{\text} is in \texttt{\list}.
\item \texttt{\_isnextchar \text \{\text\}\{\text\}} performs \texttt{\text} if next character is \texttt{\text}, else \texttt{\text}.
\item \texttt{\_kv \{\key\}} expands to value when key-value parameters are used.
\item \texttt{\_loop ... \_repeat} is classical Plain TeX loop.
\item \texttt{\_mathstyles \{\text \}} enables to create macros dependent on current math style.
\item \texttt{\_namespace \{\pkg\}} declares name space used by package writers.
\item \texttt{\newcount, \newdimen} etc. are classical Plain TeX allocators.
\item \texttt{\neowif} \texttt{\_iffoo} declares boolean \texttt{\_iffoo} as in Plain TeX.
\item \texttt{\_newfi \_iffoo} declares boolean \texttt{\_iffoo}.
\item \texttt{\_optdef \macro \{\optdefault\}\{\parameters\}\{\body\}} defines \texttt{\macro} with [opt.parameter].
\item \texttt{\_opwarning \{\text\}} prints \texttt{\text} to the terminal and .log file as warning.
\item \texttt{\_private \{\sequence\}\{\sequence\}} \{\sequence\} ... ; declares \texttt{\sequence}s for private name space.
\item \texttt{\_public \{\sequence\}\{\sequence\}} \{\sequence\} ... ; declares \texttt{\sequence}s for public name space.
\end{itemize}
\readkv \macro reads parameters from \macro in key-value format.
\replstring \macro{{\langle stringA\rangle}{\langle stringB\rangle}} replaces all \langle stringA\rangle to \langle stringB\rangle in \macro.
\sdef \{\langle string\rangle\}{\langle parameters\rangle}{\langle body\rangle} behaves like \def\{\langle string\rangle\}{\langle parameters\rangle}{\langle body\rangle}.
\settable and \restoretable manipulate with stack of catcode tables.
\slet \{\langle string\rangle\}{\langle parameters\rangle} behaves like \let\{\langle string\rangle\}={\langle parameters\rangle}.
\sdef \{\langle string\rangle\}{\langle parameters\rangle} \langle \langle body\rangle \rangle behaves like \xdef\{\langle string\rangle\}{\langle parameters\rangle} \langle \langle body\rangle \rangle.
\trycs \{\langle string\rangle\}{\langle text\rangle} expands \langle string\rangle if it is defined else expands \langle text\rangle.
\useit \{one\}, \usesecond \{one\}{two} uses given parameter.
\wlog \{\langle text\rangle\} writes \langle text\rangle to .log file.
\wterm \{\langle text\rangle\} writes \langle text\rangle to the terminal and .log file.
\xargs \{what\} \{token\} \{token\} \ldots ; repeats \{what\} \{token\} for each \{token\}.

1.10 Compatibility with Plain \TeX

All macros of Plain \TeX are re-written in Op\TeX. Common macros should work in the same sense as in original Plain \TeX. Internal control sequences like \_p@ or \_f@t are removed and mostly replaced by control sequences prefixed by \_ (like \_this). If you need to use the basic set of old Plain \TeX control sequences like \_p@ (for example you are reading an old macro file), use \load[plain-at].

All primitives and common macros have two control sequences with the same meaning: in prefixed and unprefixed form. For example \_hbox is equal to \hbox. Internal macros of Op\TeX have and use only prefixed form. User should use unprefixed forms, but prefixed forms are accessible too because the \_ is set as a letter category code globally (in macro files and users document too). Users should re-define unprefixed forms of control sequences without worries that something internal will be broken.

The Latin Modern 8bit fonts instead Computer Modern 7bit fonts are preloaded in the format, but only a few ones. The full family set is ready to use after the command \fontfam[LMfonts] which reads the fonts in OTF format.

Plain \TeX defines \newcount, \bye etc. as \outer macros. Op\TeX doesn’t set any macro as \outer. Macros like \TeX, \rm are defined as \protected.

The text accents macros \" , \', \v, \u, \^, \textaccent4, \_ are undefined in Op\TeX. Use real letters like á, ř, ž in your source document instead of these old accents macros. If you really want to use them, you can initialize them by the \oldaccents command. But we don’t recommend it.

The default paper size is not set as the letter with 1in margins but as A4 with 2.5cm margins. You can change it, for example by \margins/1 letter (1,1,1,1)in. This example sets the classical Plain \TeX page layout.

The origin for the typographical area is not at the top left 1in 1in coordinates but at the top left paper corner exactly. For example, \hoffset includes directly left margin.

The tabbing macros \settabs and \+ (from Plain \TeX) are not defined in Op\TeX because they are obsolete. But you can use the Op\TeX trick 0021 if you really need such feature.

The \sec macro is reserved for sections but original Plain \TeX declares this control sequence for math secant.\footnote{The math accents macros like \\acute, \bar, \dot, \hat still work.}

\footnote{Use $\secant(x)$ to get sec(\(x\)).}
Chapter 2

Technical documentation

This documentation is written in the source files *.opm between the _doc and _cod pairs or after the _endcode command. When the format is generated by

\texttt{luatex -ini optex.ini}

then the text of the documentation is ignored and the format optex.fmt is generated. On the other hand, if you run

\texttt{optex optex-doc.tex}

then the same *.opm files are read when the second chapter of this documentation is printed.

A knowledge about \TeX is expected from the reader. You can see a short document \TeX in a Nutshell or more detail \TeX by topic.

Notices about hyperlinks. If a control sequence is printed in red color in this documentation then this denotes its “main documentation point”. Typically, the listing where the control sequence is declared follows immediately. If a control sequence is printed in the blue color in the listing or in the text then it is an active link that points (usually) to the main documentation point. The main documentation point can be an active link that points to a previous text where the control sequence was mentioned. Such occurrences are active links to the main documentation point.

2.1 The main initialization file

The optex.ini file is read as the main file when the format is generated.

\begin{verbatim}
%% This is part of the OpTeX project, see http://petr.olsak.net/optex
%% OpTeX ini file
%% Petr Olsak <project started from: Jan. 2020>

\catcode `\{=1 % left brace is begin-group character
\catcode `\}=2 % right brace is end-group character
\catcode `\$=3 % dollar sign is math shift
\catcode `\&=4 % ampersand is alignment tab
\catcode `\#=6 % hash mark is macro parameter character
\catcode `\^^K=7 % circumflex and uparrow are for superscripts
\catcode `\^^A=8 % downarrow is for subscripts
\catcode `\"=7 % single quotes are for normal text
\catcode `\%=7 % percent sign is for comments
\catcode `\*=7 % asterisk is for comments
\catcode `\^=7 % caret is for comments
\catcode `\_%=7 % underscore is for comments
\catcode `\~%=7 % tilde is for comments
\catcode `\^[=7 % backtick is for comments
\\def\optexversion{1.05 Jan.2022}
\def\fmtname{OpTeX}
\let\fmtversion=\optexversion
\end{verbatim}

Category codes are set first. Note that the _ is set to category code “letter”, it can be used as a part of control sequence names. Other category codes are set as in plain \TeX. The \texttt{\optexversion} and \texttt{\fmtname} are defined.

\begin{verbatim}
\def\optexversion{1.05 Jan.2022}
\def\fmtname{OpTeX}
\let\fmtversion=\optexversion
\end{verbatim}

We check if Lua\TeX engine is used at -ini state. And the \texttt{^J} character is set as \texttt{\newlinechar}.  

30
The basic macros for macro file syntax is defined, i.e. \_endcode, \_doc and \_cod. The \_codedecl will be re-defined later.

Individual *.opm* macro files are read.

The file *optex.lua* is embedded into the format as byte-code. It is documented in section 2.39.
The \everyjob register is initialized and the format is saved by the \dump command.

\everyjob = { %
  \message{This is OpTeX (Olsak's Plain TeX), version \optexversion}
  \directlua{lua.bytecode[1]()}% load OpTeX's Lua code
  \mathsbon % replaces \int_a^b to \int _a^b
  \inputref % inputs \jobname.ref if exists
}
\dump % You can redefine \dump if additional macros are needed. Example:
% \let\dump=\relax \input optex.ini \input mymacros \dump

2.2 Concept of namespaces of control sequences

2.2.1 Prefixing internal control sequences

All control sequences used in \mbox{\textsc{OpT_Px}} are used and defined with _ prefix. The user can be sure that when he/she does \texttt{\textbackslash def\_foo} then neither internal macros of \mbox{\textsc{OpT_Px}} nor \textsc{TeX} primitives will be damaged. For example \texttt{\textbackslash def\_if(...) will not damage macros because \mbox{\textsc{OpT_Px}}'s macros are using \_if instead of \if.}

All \textsc{TeX} primitives are initialized with two representative control sequences: \texttt{\word} and \texttt{\_word}, for example \texttt{\hbox} and \texttt{\_hbox}. The first alternative is reserved for users or such control sequences can be re-defined by a user.

\mbox{\textsc{OpT_Px}} sets the character _ as letter, so it can be used in control sequences. When a control sequence begins with this character then it means that it is a primitive or it is used in \mbox{\textsc{OpT_Px}} macros as internal. User can redefine such prefixed control sequence only if he/she explicitly knows what happens.

We never change catcode of _, so internal macros can be redefined by user without problems if it is desired. We don’t need something like \texttt{\makeatletter} from \textsc{LaTeX}.

\mbox{\textsc{OpT_Px}} defines all new macros as prefixed. For public usage of such macros, we need to set their non-prefixed versions. This is done by

\texttt{\public \langle list of control sequences \rangle ;}

For example \texttt{\public \_foo \_bar ; does \let\_foo=\_\foo, \let\_bar=\_\bar.}

At the end of each code segment in \mbox{\textsc{OpT_Px}}, the \_public macro is used. You can see which macros are defined for public usage in that code segment.

The macro \texttt{\private} does the reverse job of \texttt{\public} with the same syntax. For example \texttt{\private \_foo \_bar ; does \let\_\foo=\_foo, \let\_\bar=\_\bar.} This should be used when an unprefixed variant of a control sequence is declared already but we need the prefixed variant too.

In this documentation: if both variants of a control sequence are declared (prefixed and unprefixed), then the accompanying text mentions only the unprefixed variant. The code typically defines the prefixed variant and then the \texttt{\public} (or \_public) macro is used.

2.2.2 Namespace of control sequences for users

Users can (re)define or (re)declare any control sequence with a name without any _. This does not make any problem in internal \mbox{\textsc{OpT_Px}} macros.\footnote{The token \texttt{\par} is in user name space too from \textsc{OpT_Px} 1.04+ and \textsc{LaTeX} 1.14, see also the end of section 2.38.}

User can define or declare control sequences with _ character, for example \texttt{\_my\_control\_sequence}, but with the following exceptions:

\begin{itemize}
  \item Control sequences which begin with _ are reserved for \textsc{TeX} primitives, \mbox{\textsc{OpT_Px}} internal macros and packages internal macros.
  \item Multiletter control sequences in the form \texttt{\langle word \rangle_}, or \texttt{\langle word \rangle_\langle one-letter \rangle}, where \texttt{\langle word \rangle} is a sequence of letters, are inaccessible, because they are interpreted as \texttt{\langle word \rangle} followed by _ or as \texttt{\langle word \rangle} followed by _\langle one-letter \rangle. This is important for writing math, for example:
\end{itemize}
\int_a^b \ldots \text{ is interpreted as } \int_a^b
\max_M \ldots \text{ is interpreted as } \max_M
\alpha_{ij} \ldots \text{ is interpreted as } \alpha_{ij}

This feature is implemented using Lua code at input processor level, see the section 2.15 for more details. You can deactivate this feature by \mathsboff. After this, you can still write $\int_a^b$ (Unicode) or $\int_a^b$ without problems but \int_a^b yields to undefined control sequence \int_a. You can activate this feature again by \mathsbon. The effect will take shape from next line read from input file.

- Control sequences in the form \_⟨pkg⟩_⟨word⟩ is intended for package writers as internal macros for a package with ⟨pkg⟩ identifier, see section 2.2.4.

The single-letter control sequences like \%, \$, ^ etc. are not used in internal macros. Users can redefine them, but (of course) some classical features can be lost (printing percent character by \% for example).

### 2.2.3 Macros files syntax

Each segment of OpTEX macros is stored in one file with .opm extension (means OPtex Macros). Your local macros should be in a normal *.tex file.

The code in macro files starts by \_codedecl and ends by \_endcode. The \_endcode is equivalent for \endinput, so documentation can follow. The \_codedecl has syntax:

```
\_codedecl \sequence {Name <version>}
```

If the mentioned \sequence is defined, then \_codedecl does the same as \endinput: this protects from reading the file twice. We suppose, that \sequence is defined in the macro file.

It is possible to use the \_doc ... \_cod pair between the macro lines. The documentation text should be here. It is ignored when macros are read but it can be printed using doc.opm macros like in this documentation.

### 2.2.4 Name spaces for package writers

Package writer should use internal names in the form \_⟨pkg⟩_⟨sequence⟩, where ⟨pkg⟩ is a package label. For example: \_qr_utfstring from qrcode.opm package.

The package writer does not need to write repeatedly \_pkg_foo \_pkg_bar etc. again and again in the macro file.\footnote{We have not adopted the idea from expl3 language:}

When the \_namespace {⟨pkg⟩} is declared at the beginning of the macro file then all occurrences of \_foo will be replaced by \_⟨pkg⟩_foo at the input processor level. The macro writer can write (and backward can read his/her code) simply with \_foo, \_bar control sequences and \_⟨pkg⟩_foo, \_⟨pkg⟩_bar control sequences are processed internally. The scope of the \_namespace command ends at the \_endnamespace command or when another \_namespace is used. This command checks if the same package label is not declared by the \_namespace twice.

The \_nspublic macro does \texttt{let} \_foo = \_⟨pkg⟩_foo when \_namespace {⟨pkg⟩} is declared. Moreover, it prints a warning if \_foo is defined already. The \_nsprivate macro does reverse operation to it without warnings. Example: you can define \texttt{def \_macro{...}} and then set it to the user name space by \_nspublic \macro;

Don’t load other packages (which are using their own namespace) inside your namespace. Do load them before your \_namespace {⟨pkg⟩} is initialized. Or close your namespace by \_endnamespace and open it again (after other packages are loaded) by \_resetnamespace {⟨pkg⟩}.

If the package writer needs to declare a control sequence by \_newif, then there is an exception of the rule described above. Use \_newif\_if⟨⟨pkg⟩⟩_bar, for example \_newif\_ifqr_incorner. Then the control sequences \_qr_incornertrue and \_qr_incornerfalse can be used (or the sequences \_incornertrue and \_incornerfalse when \_namespace{qr} is used).

### 2.2.5 Summary about rules for external macro files published for OpTEX

If you are writing a macro file that is intended to be published for OpTEX, then you are greatly welcome. You should follow these rules:

\footnote{We have not adopted the idea from expl3 language:}
• Don’t use control sequences from the user namespace in the macro bodies if there is no explicit and documented reason to do this.
• Don’t declare control sequences in the user namespace if there are no explicit and documented reasons to do this.
• Use control sequences from OpTEX and primitive namespace in read-only mode, if there is not an explicit and documented reason to redefine them.
• Use `\_⟨pkg⟩_⟨name⟩` for your internal macros or `\⟨name⟩` if the `\namespace{⟨pkg⟩}` is declared. See section 2.2.4.
• Use `\load` (or better: `\_load`) for loading more external macros if you need them. Don’t use `\_input` explicitly in such cases. The reason is: the external macro file is not loaded twice if another macro or the user needs it explicitly too.
• Use `\_codedecl` as your first command in the macro file and `\_endcode` to close the text of macros.
• Use `\_doc ... \_cod` pairs for documenting the code pieces.
• You can write more documentation after the `\_endcode` command.
• The OpTEX catcodes are set when `\load` your package (i.e. plain TEX catcodes plus catcode of _ is 11). If a catcode is changed during loading your package then it is forgot because `\load` returns to catcodes used before loading package. If you want to offer a catcode changing for users then insert it to a macro which can be used after loading.

If the macro file accepts these recommendations then it should be named by ⟨filename⟩.opm where ⟨filename⟩ differs from file names used directly in OpTEX and from other published macros. This extension .opm has precedence before .tex when the `\load` macro is used.

The qrcode.opm is the first example of how an external macro file for OpTEX can look like.

2.2.6 The implementation of the namespaces

```
\_codedecl \public {Prefixing and code syntax <2021-08-16>} % preloaded in format
```

All TeX primitives have alternative control sequence `\_hbox \_string, ...

```
\let\_directlua = \directlua
\_directlua {
% enable all TeX primitives with _ prefix
\tex.enableprimitives('_', \tex.extraprimitives('tex'))
% enable all primitives without prefixing
\tedx.enableprimitives('', \tedx.extraprimitives())
% enable all primitives with _ prefix
\tedx.enableprimitives('_', \tedx.extraprimitives())
}
\ea is useful shortcut for `\expandafter`. We recommend to use always the private form of \_ea because there is high probability that \ea will be redefined by the user.
\public ⟨sequence⟩ ⟨sequence⟩ ... ; does `\let \⟨sequence⟩ = \_⟨sequence⟩` for all sequences.
\private ⟨sequence⟩ ⟨sequence⟩ ... ; does `\let \_⟨sequence⟩ = \⟨sequence⟩` for all sequences.
\_checkexists ⟨where⟩ ⟨prefix⟩⟨sequence⟩ prints error if the control sequence propagated to a new name space by \public etc. macros is not declared.
\xargs ⟨what⟩ ⟨sequence⟩ ⟨sequence⟩ ... ; does ⟨what⟩⟨sequence⟩ for each sequences.
```

```
\_let\_es = \_expandafter % usefull shortcut
\_long\_def \_xargs #1#2{\_ifx #2;\_else \_es#1\_es\_xargs \_es #1\_fi}
\_def \_pkglabel(){}
\_def \_public {\_xargs \_publicA}
\_def \_publicA #1{% 
\_checkexists \public #1%
\_es\_let \_es#1\_csname _\_csstring #1\_endcsname
}
\_def \_private {\_xargs \_privateA}
\_def \_privateA #1{% 
\_checkexists \private {}#1%
\_es\_let \_csname _\_csstring #1\_endcsname =#1%
}
\_def \_checkexists #1#2#3{% \unless \_ifcsname #2\_csstring#3\_endcsname #1#2\_csstring#3\_endcsname
```

34
Each macro file should begin with \_codedecl \macro \langle info \rangle \}. If the \macro is defined already then the \endinput protects to read such file more than once. Else the \langle info \rangle is printed to the terminal and the file is read.

The \_endcode is defined as \endinput in the optex.ini file. \vterm \{\langle text \rangle\} prints the \langle text \rangle to the terminal and to the .log file, \vlog \{\langle text \rangle\} prints the \langle text \rangle only to the .log file (as in plain \TeX).

The \optexversion and \fmtname are defined in the optex.ini file. Maybe, somebody will need a private version of these macros.

The \_mathsbon and \_mathsboff are defined in math-macros.opm file. Now, we define the macros \_namespace \{\langle pkg label \rangle\}, \_resetnamespace \{\langle pkg label \rangle\}, \_endnamespace, \_nspublic and \_nsprivate for package writers, see section 2.2.4.

2.3 pdf\TeX\ initialization

Common pdf\TeX\ primitives equivalents are declared here. Initial values are set.
\_codedecl \pdfprimitive {Lu\TeX\ initialization code <2020-02-21>} \% preloaded in format

\_let\_pdfpagewidth \pagewidth
\_let\_pdfpageheight \pageheight
\_let\_pdfadjustspacing \adjustspacing
\_let\_pdfprotrudechars \protrudechars
\_let\_pdfnoligatures \ignoreligaturesinfont
\_let\_pdffontexpand \expandglyphsinfont
\_let\_pdfcopyfont \copyfont
\_let\_pdfxform \saveboxresource
\_let\_pdflastxform \lastsavedboxresourceindex
\_let\_pdfrefxform \useboxresource
\_let\_pdfximage \saveimageresource
\_let\_pdflastximage \lastsavedimageresourceindex
\_let\_pdflastximagepages \lastsavedimageresourcepages
\_let\_pdfrefximage \useimageresource
\_let\_pdflastximagepages \lastsavedimageresourcepages
\_let\_pdfsavepos \savepos
\_let\_pdflastxpos \lastxpos
\_let\_pdflastypos \lastypos
\_let\_pdfoutput \outputmode
\_let\_pdfdraftmode \draftmode
\_let\_pdfpxdimen \pxdimen
\_let\_pdfinsertht \insertht
\_let\_pdfnormaldeviate \normaldeviate
\_let\_pdfuniformdeviate \uniformdeviate
\_let\_pdfsetrandomseed \setrandomseed
\_let\_pdfrandomseed \randomseed
\_let\_pdfprimitive \primitive
\_let\_ifpdfprimitive \ifprimitive
\_let\_ifpdfabsnum \ifabsnum
\_let\_ifpdfabsdim \ifabsdim

\directlua {tex.enableprimitives('pdf',{'tracingfonts'})}

\_protected\_def \_pdftexversion {\_numexpr 140\_relax}
\_def \_pdftexrevision {7}
\_protected\_def \_pdflastlink {\_numexpr \pdffeedback lastlink\_relax}
\_protected\_def \_pdfretval {\_numexpr \pdffeedback retval\_relax}
\_protected\_def \_pdflastobj {\_numexpr \pdffeedback lastobj\_relax}
\_protected\_def \_pdflastannot {\_numexpr \pdffeedback lastannot\_relax}
\_def \_pdfxformname {\pdffeedback xformname}
\_def \_pdfcreationdate {\pdffeedback creationdate}
\_def \_pdffontname {\pdffeedback fontname}
\_def \_pdffontobjnum {\pdffeedback fontobjnum}
\_def \_pdffontsize {\pdffeedback fontsize}
\_def \_pdfpageref {\pdffeedback pageref}
\_def \_pdfcolorstackinit {\pdffeedback colorstackinit}
\_protected\_def \_pdfliteral {\pdffeedback literal}
\_protected\_def \_pdfcolorstack {\pdffeedback colorstack}
\_protected\_def \_pdfsetmatrix {\pdffeedback setmatrix}
\_protected\_def \_pdfsave {\pdffeedback save\_relax}
\_protected\_def \_pdfrestore {\pdffeedback restore\_relax}
\_protected\_def \_pdfobj {\pdffeedback obj}
\_protected\_def \_pdffobj {\pdffeedback refobj}
\_protected\_def \_pdfannot {\pdffeedback annot}
\_protected\_def \_pdfstartlink {\pdffeedback startlink}
\_protected\_def \_pdfendlink {\pdffeedback endlink\_relax}
\_protected\_def \_pdfoutline {\pdffeedback outline}
\_protected\_def \_pdfdest {\pdffeedback dest}
\_protected\_def \_pdfthread {\pdffeedback thread}
\_protected\_def \_pdfstartthread {\pdffeedback starthread}
2.4 Basic macros

We define first bundle of basic macros.

```
\_\_protected_\_def \_pdfendthread \{\_pdfextension endthread\_relax\}
\_\_protected_\_def \_pdfinfo \{\_pdfextension info \}
\_\_protected_\_def \_pdfcatalog \{\_pdfextension catalog \}
\_\_protected_\_def \_pdfnames \{\_pdfextension names \}
\_\_protected_\_def \_pdfincludechars \{\_pdfextension includechars \}
\_\_protected_\_def \_pdffontattr \{\_pdfextension fontattr \}
\_\_protected_\_def \_pdfmapfile \{\_pdfextension mapfile \}
\_\_protected_\_def \_pdfmapline \{\_pdfextension mapline \}
\_\_protected_\_def \_pdftrailer \{\_pdfextension trailer \}
\_\_protected_\_def \_pdfglyphtounicode \{\_pdfextension glyphtounicode \}
\_\_protected_\_def \_pdfcompresslevel \{\_pdfvariable compresslevel\}
\_\_protected_\_def \_pdfobjcompresslevel \{\_pdfvariable objcompresslevel\}
\_\_protected_\_def \_pdfdecimaldigits \{\_pdfvariable decimaldigits\}
\_\_protected_\_def \_pdfgamma \{\_pdfvariable gamma\}
\_\_protected_\_def \_pdfimageresolution \{\_pdfvariable imageresolution\}
\_\_protected_\_def \_pdfimagegamma \{\_pdfvariable imagegamma\}
\_\_protected_\_def \_pdfimagehicolor \{\_pdfvariable imagehicolor\}
\_\_protected_\_def \_pdfimageaddfilename \{\_pdfvariable imageaddfilename\}
\_\_protected_\_def \_pdfpkresolution \{\_pdfvariable pkresolution\}
\_\_protected_\_def \_pdfinclusioncopyfonts \{\_pdfvariable inclusioncopyfonts\}
\_\_protected_\_def \_pdfinclusionerrorlevel \{\_pdfvariable inclusionerrorlevel\}
\_\_protected_\_def \_pdfgentounicode \{\_pdfvariable gentounicode\}
\_\_protected_\_def \_pdfminorversion \{\_pdfvariable minorversion\}
\_\_protected_\_def \_pdfuniqueresname \{\_pdfvariable uniqueresname\}
\_\_protected_\_def \_pdfhorigin \{\_pdfvariable horigin\}
\_\_protected_\_def \_pdfvorigin \{\_pdfvariable vorigin\}
\_\_protected_\_def \_pdflinkmargin \{\_pdfvariable linkmargin\}
\_\_protected_\_def \_pdfdestmargin \{\_pdfvariable destmargin\}
\_\_protected_\_def \_pdfthreadmargin \{\_pdfvariable threadmargin\}
\_\_protected_\_def \_pdfpagesattr \{\_pdfvariable pagesattr\}
\_\_protected_\_def \_pdfpageattr \{\_pdfvariable pageattr\}
\_\_protected_\_def \_pdfpageresources \{\_pdfvariable pageresources\}
\_\_protected_\_def \_pdfxformattr \{\_pdfvariable xformattr\}
\_\_protected_\_def \_pdfxformresources \{\_pdfvariable xformresources\}
\_\_protected_\_def \_pdfpkmode \{\_pdfvariable pkmode\}
```

```
\public
\pdftexversion \pdftexrevision \pdflastlink \pdfreval \pdflastobj
\pdflastannot \pdfsformname \pdfcreationdate \pdffontname \pdffontobjnum
\pdffontsize \pdfpagemark \pdfcolorstackinit \pdfliteral \pdfcolorstack
\pdfsetmatrix \pdfsave \pdfrestore \pdfobj \pdfreffobj \pdfannot
\pdfstartlink \pdfendlink \pdfoutline \pdfdest \pdfthread \pdfstartthread
\pdfendthread \pdfinfo \pdfcatalog \pdfnames \pdfincludechars \pdffontattr
\pdfmapfile \pdfmapline \pdftrailer \pdfglyphtounicode \pdfcompresslevel
\pdfobjcompresslevel \pdfdecimaldigits \pdfgamma \pdfimageresolution
\pdfimagegamma \pdfimagehicolor \pdfimageaddfilename
\pdfxformattr \pdfinclusioncopyfonts \pdfinclusionerrorlevel
\pdfgentounicode \pdfpagebox \pdfminorversion \pdfuniqueresname \pdfhorigin
\pdfvorigin \pdflinkmargin \pdfdestmargin \pdfthreadmargin \pdfpageattr
\pdfpagesattr \pdfpageresources \pdfxformresources \pdfpkmode
```

```
2.4 Basic macros

We define first bundle of basic macros.

```
__codedcl \def \{Basic macros for \TeX<2021-07-20>\} % preloaded in format
\bgroup, \egroup, \empty, \space, and \null are classical macros from plain \TeX.
```

37
\\_let\_bgroup={ \_let \_egroup=}
\def \_empty {}
\def \_space { }
\def \_null { \_hbox{}}
\public \bgroup \egroup \empty \space \null ;

\_ignoreit ignores next token or \{\text{⟨text⟩}\} \{\text{⟨text⟩}\} expands to \langle\text{⟨text⟩}\} (removes outer braces), \_ignoresecond uses first, ignores second parameter and \_usesecond ignores first, uses second parameter.

\_long\_def \_ignoreit #1{}
\_long\_def \_useit #1{#1}
\_long\_def \_ignoresecond #1#2{#1}
\_long\_def \_usesecond #1#2{#2}
\public \ignoreit \useit \ignoresecond \usesecond ;

\bslash is “normal backslash” with category code 12. \nbb is double backslash and \pcent is normal %.

They can be used in Lua codes, for example.

\_edef \_bslash {\_csstring\}
\_edef \_nbb {\_bslash\_bslash}
\_edef \_pcent {\_csstring\%}
\_public \bslash \nbb \pcent ;

\sdef {\langle\text⟩}\{\langle\text⟩\} is equivalent to \sdef\langle\text⟩, where \langle\text⟩ is a control sequence. You can use arbitrary parameter mask after \sdef\langle\text⟩, don’t put the (unwanted) space immediately after closing brace \}.

\sxdef {\langle\text⟩}\{\langle\text⟩\} is equivalent to \sxdef\langle\text⟩.

\slet {\langle\textA⟩}{\langle\textB⟩} is equivalent to \slet\langle\textA⟩=\langle\textB⟩.

\_def \_sdef #1\{\_ea\_def \_csname#1\_endcsname}
\_def \_sxdef #1\{\_ea\_xdef \_csname#1\_endcsname}
\_def \_slet #1#2\{\_ea\_let \_csname#1\_ea\_endcsname \_ifcsname#2\_ea\_endcsname \_begincsname#2\_endcsname \_else \_undefined \_fi
\_public \sdef \sxdef \slet ;

\adef {\langle\text⟩}{\langle\text⟩} puts the \langle\text⟩ as active character and defines it as \{\langle\text⟩\}. You can declare a macro with parameters too. For example \adef #\{\ldots#\ldots\}.

\cs {\langle\text⟩}\{\langle\text⟩\endcsname} is only a shortcut to \csname\langle\text⟩\endcsname, but you need one more \_ea if you need to get the real control sequence \langle\text⟩.

\trycs {\langle\text⟩}\{\langle\text⟩\} expands to \{\langle\text⟩\} if it is defined else to the \langle\text⟩.

\addto \macro{\langle\text⟩}\{\langle\text⟩\} adds \langle\text⟩ to your \macro, which must be defined.

\incr \langle\text⟩ increases \langle\text⟩ by one globally. \decr \langle\text⟩ decreases \langle\text⟩ by one globally.

\opwarning {\langle\text⟩} prints warning on the terminal and to the log file.

\loggingall and \tracingall are defined similarly as in plain \TeX, but they print more logging information to the log file and the terminal.
\_def\_loggingall{\_tracingcommands=3 \_tracingstats=2 \_tracingpages=1 \_tracingoutput=1 \_tracinglostchars=1 \_tracingmacros=3 \_tracingparagraphs=1 \_tracingrestores=1 \_tracingscntokens=1 \_tracingifs=1 \_tracinggroups=1 \_tracingassigns=1}
\_def\_tracingall{\_tracingonline=1 \_loggingall}
\_public \_loggingall \_tracingall ;

\_byehook is used in the \_bye macro. Write a warning if the user did not load a Unicode Font. Write a “rerun” warning if the .ref file was newly created or it was changed (compared to the previous TeX run).

122 \_def\_byehook{%
123 \_ifx\_initunifonts\_relax \_relax\_else \_opwarning{Unicode font was not loaded}\_fi
124 \_immediate\_closeout\_reffile
125 \_edef\_tmp{\_mdfive{\_jobname.ref}}%
126 \_ifx\_tmp\_prevrefhash\_else \_opwarning{Try to rerun, \_jobname.ref file was \_ifx\_prevrefhash\_empty created\_else changed}\_fi\_fi
}

2.5 Allocators for TeX registers

Like plainTeX, the allocators \newcount, \newwrite, etc. are defined. The registers are allocated from 256 to the \_mai\langle type\rangle which is 65535 in LuaTeX.
Unlike in PlainTeX, the mentioned allocators are not \outer.
User can use \dimen0 to \dimen200 and similarly for \skip, \muskip, \box, and \toks directly.
User can use \count20 to \count200 directly too. This is the same philosophy as in old plainTeX, but the range of directly used registers is wider.
Inserts are allocated from 254 to 201 using \newinsert.
You can define your own allocation concept (for example for allocation of arrays) from the top of the registers array. The example shows a definition of the array-like declarator of counters.
\newcount \_maicount % redefine maximal allocation index as variable
\_maicount = \maicount % first value is top of the array
\def\newcountarray #1[#2]{% \newcountarray \foo[100]
  \global\advance\_maicount by -#2\relax
  \ifnum \_countalloc > \_maicount
    \errmessage{No room for a new array of \string\count}%
  \else
    \global\chardef#1=\_maicount
  \fi
}
\def\usecount #1[#2]{% \usecount \foo[2]
  \count\numexpr#1+#2\relax
}

The limits are set first.
\_chardef\_maicount = 65535 \_chardef\_maimain = 65535 \_let\_maimaindimen = \_maicount
\_let\_maimain = \_maicount
\_let\_maimainskip = \_maicount
\_let\_maimainmuskip = \_maicount
\_let\_maimainbox = \_maicount
\_let\_maimaintoks = \_maicount
\_chardef\_mairead = 15
\_chardef\_maiwrite = 15
\_chardef\_maifam = 255

Each allocation macro needs its own counter.
The common allocation macro \_allocator \langle sequence \rangle \{\langle type \rangle \} \langle primitive declarator \rangle is defined. This idea was used in classical plain TEX by Donald Knuth too but the macro from plain TEX seems to be more complicated:)

\def\_allocator #1\#2\#3{
\_incr{\cs{\#2alloc}}
\_ifnum\cs{\#2alloc} > \cs{\mai\#2}\
\_errmessage{No room for a new \_ea\_string\_csname \#2\_endcsname}\
\_else
\_global\#3\#1=\cs{\#2alloc}
\_wlog{\_string#1=\_ea\_string\_csname \#2\_endcsname\_the\_cs{\#2alloc}}
\fi
}

The allocation macros \newcount, \newdimen, \newskip, \newmuskip, \newbox, \newtoks, \newread, \newwrite and \newfam are defined here.

\def\newcount \#1{\_allocator \#1{count}\_countdef}
\def\newdimen \#1{\_allocator \#1{dimen}\_dimendef}
\def\newskip \#1{\_allocator \#1{skip}\_skipdef}
\def\newmuskip \#1{\_allocator \#1{muskip}\_muskipdef}
\def\newbox \#1{\_allocator \#1{box}\_chardef}
\def\newtoks \#1{\_allocator \#1{toks}\_toksdef}
\def\newread \#1{\_allocator \#1{read}\_chardef}
\def\newwrite \#1{\_allocator \#1{write}\_chardef}
\def\newfam \#1{\_allocator \#1{fam}\_chardef}

\public \newcount \newdimen \newskip \newmuskip \newbox \newtoks \newread \newwrite \newfam ;

The \newinsert macro is defined differently than others.

\def\newinsert \#1{\_Allocator \#1{insert}\_insertdef}
\def\newinsertmin = 201
\_decr\insertalloc
\_ifnum\insertalloc < \insertmin\
\_errmessage{No room for a new \_string\_insert}\%
\else
\_global\chardef#1=\insertalloc
\_wlog{\_string#1=\_string\_insert\_the\_insertalloc}\%
\fi

\public \newinsert ;

Other allocation macros \newattribute and \newcatcodetable have their counter allocated by the \newcount macro.

\def\newattribute \#1{\_Allocator \#1{attribute}\_attributedef}
\def\newcatcodetable \#1{\_Allocator \#1{catcodetable}\_chardef}

\public \newattribute \newcatcodetable ;

We declare public and private versions of \tmpnum and \tmpdim registers separately. They are independent registers.
A few registers are initialized like in plain \TeX. We absolutely don’t support the \@category dance, so \z@skip, \z@, \@ etc. are not defined in Op\TeX. If you need such control sequences then you can initialize them by \load[plain-at].

Only the \zo and \zoskip (equivalents to \z@ and \z@skip) are declared here and used in some internal macros of Op\TeX for improving speed.

\begin{verbatim}
122 _newdimen _maxdimen _maxdimen=16383.99999pt % the largest legal <dimen>
123 _newdimen _zo _zo=0pt
124 _newskip _hideskip _hideskip=-1000pt plus 1fill % negative but can grow
125 _newskip _centering _centering=0pt plus 1000pt minus 1000pt
126 _newskip _zoskip _zoskip=0pt plus0pt minus0pt
127 _newbox _voidbox % permanently void box register
128 \_public _maxdimen _hideskip _centering _voidbox ;
\end{verbatim}

2.6 If-macros, loops, is-macros

2.6.1 Classical \newif

The \newif macro implements boolean value. It works as in plain \TeX. It means that after \newif\ifxxx you can use \xxttrue or \xxfalse to set the boolean value and use \ifxxx true\else false\fi to test this value. The default value is false.

The macro \newifi enables to declare \_ifxxx and to use \_xxttrue and \_xxfalse. This means that it is usable for the internal namespace (_prefixed macros).

\begin{verbatim}
18 \def \newif #1{\_es\newifi \_string #1\_relax#1}
19 \esdef \_es\newifi \_string\if #1\_relax#2[\
20 \_sdef{\_#1true}{\_let#2=\_iftrue}\
21 \_sdef{\_#1false}{\_let#2=\_iffalse}\
22 \_let#2=\_iffalse
23 }
24 \def \newifi #1{\_es\newifi \_string\_if #1\_relax#1}
25 \esdef \_es\newifi \_string\_if #1\_relax#2[\
26 \_sdef{\_#1true}{\_let#2=\_iftrue}\
27 \_sdef{\_#1false}{\_let#2=\_iffalse}\
28 \_let#2=\_iffalse
29 }
30 \_public \newif ;
\end{verbatim}

\afterfi {⟨what to do⟩}⟨ignored⟩\fi closes condition by \fi and processes ⟨what to do⟩. Usage:

\begin{verbatim}
\if<something> \afterfi{⟨result is true⟩} \else \afterfi{⟨result is false⟩} \fi
\end{verbatim}

2.6.2 Loops

The \loop ⟨codeA⟩ \ifsomething ⟨codeB⟩ \repeat loops ⟨codeA⟩⟨codeB⟩ until \ifsomething is false. Then ⟨codeB⟩ is not executed and loop is finished. This works like in plain \TeX, but implementation is somewhat better (you can use \else clause after the \ifsomething).

There are public version \loop... \repeat and private version \loop... \_repeat. You cannot mix both versions in one loop.

The \loop macro keeps its original plain \TeX meaning. It is not expandable and nested \loops are possible only in a \TeX group.
\foreach \langle list \rangle \do \{ \langle what \rangle \} repeats \langle what \rangle for each element of the \langle list \rangle. The \langle what \rangle can include \#1 which is substituted by each element of the \langle list \rangle. The macro is expandable.

\foreach \langle list \rangle \do \{ \langle parameter-mask \rangle \langle what \rangle \} reads parameters from \langle list \rangle repeatedly and does \langle what \rangle for each such reading. The \langle parameter-mask \rangle can be numeric expressions. The macro is expandable.

\fornum \langle from \rangle \ldots \langle to \rangle \do \{ \langle what \rangle \} or \fornumstep \langle num \rangle: \langle from \rangle \ldots \langle to \rangle \do \{ \langle what \rangle \} repeats \langle what \rangle for each number from \langle from \rangle to \langle to \rangle (with step \langle num \rangle or with step one). The \langle what \rangle can include \#1 which is substituted by current number. The \langle from \rangle, \langle to \rangle, \langle step \rangle parameters can be numeric expressions. The macro is expandable.

The test in the \_fornumB says: if (\langle to \rangle < \langle current number \rangle AND \langle step \rangle is positive) or if (\langle to \rangle > \langle current number \rangle AND \langle step \rangle is negative) then close loop by \_getforstack. Sorry, the condition is written by somewhat cryptoid \TeX{} language.
\for* macro ends. The \_forlevel variable keeps the current nesting level. If it is zero, then we need not save nor restore any data.

User can define own expandable “foreach” macro by \foreachdef \macro \{parameter-mask\} \{\langle what\rangle\} which can be used by \macro \{\langle list\rangle\}. The macro reads repeatedly parameters from \langle list\rangle using \langle parameter-mask\rangle and does \langle what\rangle for each such reading. For example

\foreachdef\mymacro #1,{{[#1]}}
\mymacro{a,b,cd,efg,}

expands to [a][b][cd][efg]. Such user defined macros are more effective during processing than \foreach itself because they need not to operate with the for-stack.

2.6.3 Is-macros

There are a collection of macros \isempty, \istoksempty, \isequal, \ismacro, \isdefined, \isinlist, \isfile and \isfont with common syntax:

\issomething \langle params\rangle \iftrue \langle codeA\rangle \else \langle codeB\rangle \fi

or

\issomething \langle params\rangle \iffalse \langle codeB\rangle \else \langle codeA\rangle \fi

The \else part is optional. The \langle codeA\rangle is processed if \issomething\langle params\rangle generates true condition. The \langle codeB\rangle is processed if \issomething\langle params\rangle generates false condition.

The \iftrue or \iffalse is an integral part of this syntax because we need to keep skippable nested \if conditions.

Implementation note: we read this \iftrue or \iffalse into unseparated parameter and repeat it because we need to remove an optional space before this command.

\isempty \langle\langle text\rangle\rangle \iftrue is true if the \langle text\rangle is empty. This macro is expandable.
\istoksempty \langle tokens variable\rangle \iftrue is true if the \langle tokens variable\rangle is empty. It is expandable.

\isequal \langle textA\rangle \{\langle textB\rangle\} \iftrue is true if the \langle textA\rangle and \langle textB\rangle are equal, only from strings point of view, category codes are ignored. The macro is expandable.
\ismacro \macro{\text}{\textbf{iftrue}} is true if macro is defined as ⟨\text⟩. Category codes are ignored in this testing. The macro is expandable.

\isdefined ⟨\csname⟩\textbf{iftrue} is true if ⟨\csname⟩ is defined. The macro is expandable.

\isinlist \list{⟨\text⟩}\textbf{iftrue} is true if the ⟨\text⟩ is included the macro body of the \list. The category codes are relevant here. The macro is not expandable.

\isfile ⟨\filename⟩\textbf{iftrue} is true if the file ⟨\filename⟩ exists and are readable by \TeX.

\isfont ⟨\fontname or \[fontfile]\⟩\textbf{iftrue} is true if a given font exists. The result of this testing is saved to the \_ifexistfam.

The last macro ⟨\isnextchar\char⟩⟨\codeA⟩⟨\codeB⟩ has a different syntax than all other is-macros. It executes ⟨\codeA⟩ if next character is equal to ⟨\char⟩. Else the ⟨\codeB⟩ is executed. The macro is not expandable.

### 2.7 Setting parameters

The behavior of document processing by \OpTeX{} is controlled by \textit{parameters}. The parameters are

- primitive registers used in build-in algorithms of \TeX{},
- registers declared and used by \OpTeX{} macros.
Both groups of registers have their type: number, dimension, skip, token list.

The registers are represented by their names (control sequences). If the user re-defines this control sequence then the appropriate register exists steadily and build-in algorithms are using it without change. But user cannot access its value in this case. Op\TeX{} declares two control sequences for each register: prefixed (private) and unprefixed (public). Op\TeX{} macros use only prefixed variants of control sequences. The user should use the unprefixed variant with the same meaning and set or read the values of registers using the unprefixed variant. If the user re-defines the unprefixed control sequence of a register then Op\TeX{} macros still work without change.

2.7.1 Primitive registers

The primitive registers with the same default value as in plain \TeX{} follow:

\begin{verbatim}
\_parindent=20pt % indentation of paragraphs
\_pretolerance=100 % parameters used in paragraph breaking algorithm
\_tolerance=200
\_hbadness=1000
\_vbadness=1000
\_doublehyphenpenalty=10000
\_finalhyphenpenalty=5000
\_adjdemerits=10000
\_uchyph=1
\_defaultthyphenchar=\`
\_defaultskewchar=-1
\_hfuzz=0.1pt
\_vfuzz=0.1pt
\_overfullrule=5pt % penalty between lines inside the paragraph
\_linepenalty=10 % when a word is br-k-en
\_hyphenpenalty=50 % when the hyphenmark is used explicitly
\_binoppenalty=700 % between binary operators in math
\_repenalty=500 % between relations in math
\_brokenpenalty=100 % after lines if they end by a broken word.
\_displaywidowpenalty=50 % before last line of paragraph if display math follows
\_predisplaypenalty=10000 % above display math
\_postdisplaypenalty=0 % below display math
\_delimitershortfall=5pt % parameter for scaling delimiters
\_nulldelimiterspace=1.2pt
\_scriptspace=0.5pt
\_maxdepth=4pt
\_splitmaxdepth=\_maxdimen
\_boxmaxdepth=\_maxdimen
\_parskip=0pt plus 1pt
\_abovedisplayskip=12pt plus 3pt minus 9pt
\_belowdisplayskip=12pt plus 3pt minus 9pt
\_parfillskip=0pt plus 1fil
\_thinmuskip=3mu
\_medmuskip=4mu plus 2mu minus 4mu
\_thickmuskip=5mu plus 5mu
\_topskip=10pt % top edge of page-box to first baseline distance
\_splittopskip=10pt
\end{verbatim}

Note that \_topskip and \_splittopskip are changed when first \_typosize sets the main values (default font size and default \_baselineskip).

2.7.2 Plain \TeX{} registers

Allocate registers that are used just like in plain \TeX{.}
\% We also define special registers that function like parameters:

\newskip\smallskipamount \smallskipamount=3pt plus 1pt minus 1pt
\newskip\medskipamount \medskipamount=6pt plus 2pt minus 2pt
\newskip\bigskipamount \bigskipamount=12pt plus 4pt minus 4pt
\newskip\normalbaselineskip \normalbaselineskip=12pt
\newskip\normallineskip \normallineskip=1pt
\newdimen\normallineskiplimit \normallineskiplimit=0pt
\newdimen\jot \jot=3pt
\newcount\interdisplaylinepenalty \interdisplaylinepenalty=100
\newcount\interfootnotelinepenalty \interfootnotelinepenalty=100

\def\normalbaselines{\lineskip=\normallineskip
\baselineskip=\normalbaselineskip \lineskiplimit=\normallineskiplimit}

\def\frenchspacing{\sfcode`.=1000 \sfcode`?=1000 \sfcode`!=1000
\sfcode`:=1000 \sfcode`;=1000 \sfcode`,=1000 }

\def\nonfrenchspacing{\sfcode`.=3000 \sfcode`?=3000 \sfcode`!=3000
\sfcode`:=2000 \sfcode`;=1500 \sfcode`,=1250 }

\public \normalbaselines \frenchspacing \nonfrenchspacing
\smallskipamount \medskipamount \bigskipamount
\normalbaselineskip \normallineskip \normallineskiplimit
\jot \interdisplaylinepenalty \interfootnotelinepenalty
;

\subsection*{2.7.3 Different settings than in plain \TeX}

Default “baseline setting” is for 10pt fonts (like in plain \TeX). But \texttt{\textsize} and \texttt{\textsize} macros re-declare it if another font size is used.

The \texttt{\textsize} is not set by default because the author of Op\TeX is living in Europe. If you set \texttt{\textsize} hyphenation patterns then \texttt{\textsize} is set.

The following primitive registers have different values than in plain \TeX. We prohibit orphans, set more information for tracing boxes, set page origin to the upper left corner of the paper (no at 1in, 1in coordinates) and set default page dimensions as A4, not letter.

\def\normalbaselines\% baseline setting, 10 pt font size
\emergencystretch=20pt \% we want to use third pass of paragraph building algorithm
\clubpenalty=10000 \% after first line of paragraph
\widowpenalty=10000 \% before last line of paragraph
\showboxbreadth=150 \% for tracing boxes
\showboxdepth=7
\errorcontextlines=15
\tracinglostchars=2 \% missing character warnings on terminal too

\outputmode=1 \% PDF output
\pdforigin=0pt \% origin is exactly at upper left corner
\hoffset=25mm \% margins are 2.5cm, no 1in
\voffset=25mm
\hsize=160mm \% 210mm (from A4 size) - 2*25mm (default margins)
\vsize=244mm \% 297mm (from A4 size) - 2*25mm (default margins) -3mm baseline correction
\pagewidth=210 true mm
\pageheight=297 true mm

If you insist on plain \TeX values of these parameters then you can call the \texttt{\textsize} macro.
2.7.4 OpTEX parameters

The main principle of how to configure OpTEX is not to use only parameters. A designer can copy macros from OpTEX and re-define them as required. This is a reason why we don't implement dozens of parameters, but we keep OpTEX macros relatively simple. Example: do you want another design of section titles? Copy macros \_printsec and \_printsecc from sections.opm file to your macro file and re-define them.

Notice for OPmac users: there is an important difference: all "string-like" parameters are token lists in OpTEX (OPmac uses macros for them). The reason of this difference: if a user sets parameter by unprefixed (public) control sequence, an OpTEX macro can read the same data using a prefixed (private) control sequence.

The \picdir tokens list can include a directory where image files (loaded by \inspic) are saved. Empty \picdir (default value) means that image files are in the current directory (or somewhere in the \TeX{} system where Lua\TeX{} can find them). If you set a non-empty value to the \picdir, then it must end by / character, for example \picdir={img/} means that there exists a directory img in your current directory and the image files are stored here.

You can control the dimensions of included images by the parameters \picwidth (which is equivalent to \pic) and \picheight. By default these parameters are set to zero: the native dimension of the image is used. If only \picwidth has a nonzero value, then this is the width of the image (height is calculated automatically in order to respect the aspect of the image). If only \picheight has a nonzero value then the height is given, the width is calculated. If both parameters are non-zero, the height and width are given and the aspect ratio of the image is (probably) broken. We recommend setting these parameters locally in the group where \inspic is used in order to not influence the dimensions of other images. But there exist many situations you need to put the same dimensions to more images, so you can set this parameter only once before more \inspic macros.

The \everytt is the token list used in \begtt...\endtt environment and in the verbatim group opened by \verbinput macro. You can include a code which is processed inside the group after basic settings were done. On the other hand, it is processed before the scanner of verbatim text is started. Your macros should influence scanner (catcode settings) or printing process of the verbatim code or both.

The code from the line immediately after \begtt is processed after the \everytt. This code should overwrite \everytt settings. Use \everytt for all verbatim environments in your document and use a code after \begtt locally only for this environment.

The \everyintt token list does similar work but acts in the in-line verbatim text processed by a pair of \verbchar characters or by \texttt{⟨text⟩}. You can set \everyintt={\Red} for example if you want in-line verbatim in red color.

The \ttline is used in \begtt...\endtt environment or in the code printed by \verbinput. If \ttline is positive or zero, then the verbatim code has numbered lines from \ttline+1. The \ttline register is re-set to a new value after a code piece is printed, so next code pieces have numbered lines continuously. If \ttline=-1, then \begtt...\endtt lines are without numbers and \verbinput lines show the line numbers of inputted file. If \ttline<-1 then no line numbers are printed.
The **\ttindent** gives default indentation of verbatim lines printed by **\begtt...\endtt** pair or by **\verbinput**.

The **\ttshift** gives the amount of shift of all verbatim lines to the right. Despite the **\ttindent**, it does not shift the line numbers, only the text.

The **\iindent** gives default indentations used in the table of contents, captions, lists, bib references. It is strongly recommended to re-set this value if you set **\parindent** to another value than plain \TeX default 20pt. A well-typeset document should have the same dimension for all indentations, so you should say **\ttindent=\parindent** and **\iindent=\parindent**.

The tabulator ^^I has its category code like space: it behaves as a space in normal text. This is a common \TeX setting. But in the multiline verbatim environment it is active and expands to the \hskip ⟨dimen⟩ where ⟨dimen⟩ is the width of \tabspaces spaces. Default \tabspaces=3 means that tabulator behaves like three spaces in multiline verbatim.

If **\hicolors** is non-empty then its contents is used instead **\hicolors⟨name⟩** declared in the file hisyntax-⟨name⟩.opm. The user can give his/her preferences about colors for syntax highlighting by this tokens list. The full color set must be declared here.

The default item mark used between **\begitems** and **\enditems** is the bullet. The **\defaultitem** tokens list declares this default item mark.

The **\everyitem** tokens list is applied in vertical mode at the start of each item.

The **\everylist** tokens list is applied after the group is opened by **\begitems**. The **\ilevel** keeps the value of the current nesting level of the items list.

The **\listskipamount** gives vertical skip above and below the items list if **\ilevel=1**.

The **\tit** macro includes **\vglue\titskip** above the title of the document.

The **\begmulti** and **\endmulti** pair creates more columns. The parameter **\colsep** declares the space between columns. If \num columns are specified then we have \num−1 \colseps and \num columns in total \hsize. This gives the definite result of the width of the columns.

Each line in the Table of contents is printed in a group. The **\everytocline** tokens list is processed here before the internal **\tocln:⟨num⟩** macro which starts printing the line.

The **\bibtexhook** tokens list is used inside the group when **\usebib** command is processed after style file is loaded and before printing bib-entries. You can re-define a behavior of the style file here or you can modify the more declaration for printing (fonts, baselineskip, etc.) or you can define specific macros.
used in your .bib file.
The \biboptions is used in the iso690 bib-style for global options, see section 2.32.5.
The \bibpart saves the name of bib-list if there are more bib-lists in single document, see section 2.32.1.
\everycapitonf is used before printing caption in figures and \everycapitont is used before printing
caption in tables.
\everyii tokens list is used before \noindent for each Index item when printing the Index.
The \mnotesize is the horizontal size of the marginal notes.
The \mnoteindent is horizontal space between body-text and marginal note.
\eqalign macro can be configured by \eqalign and \eqstyle tokens lists. The default values
are set in order these macro behaves like in Plain \TeX. The \eqspace is horizontal space put between
equation systems if more columns in \eqalign are used.
\lmfil is "left matrix filler" (for \matrix columns). The default value does centering because the right matrix filler is directly set to \hfil.

\\newtoks \_lmfil \_lmfil ={\_hfil}
\\public \lmfil ;

The output routine uses token lists \headline and \footline in the same sense as plain \TeX\ does. If they are non-empty then \hfil or \hss must be here because they are used inside \hbox to\hsize.

Assume that page-body text can be typeset in different sizes and different fonts and we don’t know in what font context the output routine is invoked. So, it is strongly recommended to declare fixed variants of fonts at the beginning of your document. For example \fontdef\rmfixed{\rm}, \fontdef\itfixed{\it}.

\\headline={\itfixed Text of headline, section: \fistmark \hss}
\\footline={\rmfixed \ifodd\pageno \hfill\fi \folio \hfil}

\\newdimen \_headlinedist \_headlinedist =14pt
\\newdimen \_footlinedist \_footlinedist =24pt
\\public \headlinedist \footlinedist ;

The distance between the \headline and the top of the page text is controlled by the \headlinedist register. The distance between the bottom of page-text and \footline is \footlinedist. More precisely: baseline of headline and baseline of the first line in page-text have distance \headlinedist+\topskip. The baseline of the last line in page-text and the baseline of the footline have distance \footlinedist. Default values are inspired by plain \TeX.

The \pgbottomskip is inserted to the page bottom in the output routine. You can set less tolerance here than \raggedbottom does. By default, no tolerance is given.

\\newskip \_pgbottomskip \_pgbottomskip =0pt \_relax
\\public \pgbottomskip ;

The \nextpages tokens list can include settings which will be used at next pages. It is processed at the end of output routine with \globaldefs=1 prefix. The \nextpages is reset to empty after processing. Example of usage:

\\headline={} \\nextpages={\headline={\rmfixed \firstmark \hfil}}

This example sets current page with empty headline, but next pages have non-empty headlines.

The \pgbackground token list can include macros which generate a vertical list. It is used as page background. The top-left corner of such \vbox is at the top-left corner of the paper. Example creates the background of all pages yellow:

\\pgbackground={\Yellow \hrule height 0pt depth\pdfpageheight width\pdfpagewidth}

The parameters used in \inoval and \incircle macros can be re-set by \ovalparams, \circleparams tokens lists. The default values (documented in the user manual) are set in the macros.
OpTeX defines “Standard OpTeX markup language”\(^3\) which lists selected commands from chapter 1 and gives their behavior when a converter from OpTeX document to HTML or Markdown or \TeX\ is used. The structure-oriented commands are selected here, but the commands which declare typographical appearance (page layout, dimensions, selected font family) are omitted. More information for such a converter should be given in \texttt{\cnvinfo{\{data\}}}. OpTeX simply ignores this but the converter can read its configuration from here. For example, a user can write:

\begin{verbatim}
\cnvinfo {type=html, \{cnv-to-html-data\}}
\cnvinfo {type=markdown, \{cnv-to-markdown-data\}}
\end{verbatim}

and the document can be processed by OpTeX to create PDF, or by a converter to create HTML, or by another converter to create Markdown.

\section*{2.8 More OpTeX macros}

The second bundle of OpTeX macros is here.

We define \texttt{\opinput{\{file name\}}} macro which does \texttt{\input{\{file name\}}} but the catcodes are set to normal catcodes (like OpTeX initializes them) and the catcodes setting is returned back to the current values when the file is read. You can use \texttt{\opinput} in any situation inside the document and you will be sure that the file is read correctly with correct catcode settings.

To achieve this, we declare \texttt{\optexcatcodes} catcode table and \texttt{\plaintexcatcodes}. They save the commonly used catcode tables. Note that \texttt{\catcodetable} is a part of Lua\TeX\ extension. The catcodetable stack is implemented by OpTeX macros. The \texttt{\setctable{\{catcode table\}}} pushes current catcode table to the stack and activates catcodes from the \{catcode table\}. The \texttt{\restorectable} returns to the saved catcodes from the catcode table stack.

The \texttt{\opinput} works inside the catcode table stack. It reads \texttt{\optexcatcodes} table and stores it to \texttt{\tmpcatcodes} table. This table is actually used during \texttt{\input} (maybe catcodes are changed here). Finally, \texttt{\_restoretable} pops the stacks and returns to the catcodes used before \texttt{\opinput} is run.

The implementation of the catcodetable stack follows.

The current catcodes are managed in the \texttt{\catcodetable0}. If the \texttt{\setctable} is used first (or at the outer level of the stack), then the \texttt{\catcodetable0} is pushed to the stack and the current table is re-set to the given \texttt{\{catcode table\}}. The numbers of these tables are stacked to the \texttt{\_ctablelist} macro. The \texttt{\restorectable} reads the last saved catcode table number from the \texttt{\_ctablelist} and uses it.

\footnote{Will be developed in 2021.}
When a special macro is defined with different catcodes then \normalcatcodes can be used at the end of such definition. The normal catcodes are restored. The macro reads catcodes from \optecatodes table and sets it to the main catcode table 0.

\public \setctable \restorectable ;

The \load [[filename-list]] loads files specified in comma separated \langle filename-list \rangle. The first space (after comma) is ignored using the trick \#1\#2,:: first parameter is unseparated. The \load macro saves information about loaded files by setting \_load: \langle filename \rangle as defined macro.

If the \_afterload macro is defined then it is run after \opiinput. The catcode setting should be here. Note that catcode setting done in the loaded file is forgotten after the \opiinput.

\public \load ;

The declarator \optdef \macro [⟨opt default⟩]⟨params⟩{⟨replacement text⟩} defines the \macro with the optional parameter followed by normal parameters declared in ⟨params⟩. The optional parameter must be used as the first first parameter in brackets [..]. If it isn’t used then ⟨opt default⟩ is taken into account. The ⟨replacement text⟩ can use \the\opt because optional parameter is saved to the \opt tokens register. Note the difference from L\TeX concept where the optional parameter is in \#1. Op\TeX uses \#1 as the first normal parameter (if declared).

The \nospaceafter ignores the following optional space at expand processor level using the negative \romanumeral trick.

The declarator \eoldef \macro #1{⟨replacement text⟩} defines a \macro which scans its parameter to the end of the current line. This is the parameter #1 which can be used in the ⟨replacement text⟩. The catcode of the \_endlinechar is reset temporarily when the parameter is scanned.

The macro defined by \eoldef cannot be used with its parameter inside other macros because the catcode dancing is not possible here. But the \bracedparam \macro{⟨parameter⟩} can be used here. The \bracedparam is a prefix that re-sets temporarily the \macro to a \macro with normal one parameter.

The \skiptoeol macro reads the text to the end of the current line and ignores it.
Nothing problematic happens if re-defined \catcode`" or \catcode`\.

The part of the macro removes the last processing or appends the next part to before first \langle textA \_tmptoks. It is saved to \langle textA \_tmptoks. The \scantoeos keeps category 7 for \^ in order to be able to use \^J as comment character which means that the next line continues.

How it works: \replstring\foo{\langle textA \rangle}{\langle textB \rangle} prepares \_replacestringsA\#1{\langle textA \rangle\ldots} and runs \_replacestringsA{\langle foo-body \rangle?\langle textA \rangle?\langle textA \rangle}. So, \#1 includes the first part of (foo-body) before first \langle textA \rangle. It is saved to \_tmptoks and \_replacestringsB is run in a loop. It finishes processing or appends the next part to \_tmptoks separated by \langle textB \rangle and continues loop. The final part of the macro removes the last ? from resulting \_tmptoks and defines a new version of the \_foo.

The \catcode primitive is redefined here. Why? There is very common cases like \catcode`\^ (something) or \catcode`\^ (number) but these characters ` or ^ can be set as active (typically by \verbchar macro). Nothing problematic happens if re-defined \catcode is used in this case.

If you really need primitive \catcode then you can use \catcode.

The \removespaces (text with spaces) expands to (textwithoutspaces).

The \_ea\_ignorept command expands to a decimal number \_the\(\dimen\) but without pt unit.

\public \removespaces ;
You can use expandable \bp{\dimen} convertor from TeX \dimen{} (or from an expression accepted by \dimexpr primitive) to a decimal value in big points (used as natural unit in the PDF format). So, you can write, for example:

\pdfliteral{q \_bp{.3\hsize-2mm} \_bp{2mm} m 0 \_bp{-4mm} l S Q}

You can use expandable \expr{} for analogical purposes. It expands to the value of the \expression{} at expand processor level with \_decdigits digits after the decimal point. The \expression{} can include +*() and decimal numbers in common syntax.

The usage of prefixed versions \_expr{} or \_bp{} is more recommended because a user can re-define the control sequences \expr{} or \bp{}.

\_def\_decdigits{3} % digits after decimal point in \_bp{} and \_expr{} outputs.
\_def\_pttopb{\_directlua{tex.print(string.format('\_pcent.\_decdigits f', token.scan_dimen()/65781.76))}}% pt to bp conversion
\def\_bp#1{\_ea\_pttopb\_dimexpr#1\_relax}
\def\_expr#1{\_directlua{tex.print(string.format('\_pcent.\_decdigits f',#1))}}

The pair \_doc ... \_cod{} is used for documenting macros and to printing the technical documentation of the OpTeX. The syntax is:

\_doc{} % documentation
\_cod{} % ignored text

The \documentation{} (and \ignored text too) must be \balanced text. It means that you cannot document only the { but you must document the } too.

\long\def\_doc #1\_cod {\_skiptoeol}

2.9 Using key=value format in parameters

Users or macro programmers can define macros with options in key=value format. It means a comma-separated list of equations key=value. First, we give an example.

Suppose that you want to define a macro \myframe{} with options: color of rules, color of text inside the frame, rule-width, space between text and rules. You want to use this macro as:

\myframe{}[margins=5pt,rule-width=2pt,frame-color=\Red,text-color=\Blue]{text1}

\myframe{}[frame-color=\Blue]{text2} % other parameters are default

You can define \myframe{} as follows:

\def\myframedefaults{% defaults:
  frame-color=\Black, % color of frame rules
  text-color=\Black, % color of text inside the frame
  rule-width=0.4pt, % width of rules used in the frame
  margins=2pt, % space between text inside and rules.
}
\optdef\myframe [] #1{%\bgroup
  \ea\addto\ea\myframedefaults\ea{\ea,\the\opt}%
  \readkv\myframedefaults
  \rulewidthkv{\rule-width}
  \hhkern=\kv{margins}\vvkern=\kv{margins}\relax
  \kv{frame-color}\frame{\kv{text-color}\strut #1}%
}\egroup}

We recommend using \optdef{} for defining macros with optional parameters written in []. Then the optional parameters are saved in the \opt{} tokens register. First: we append the \opt{} (actual optional parameters) to \myframedefault{} by \addto macro. Second: we read the parameters by
\readkv{parameters list} macro. Third: the values can be used by expandable \kv{key} macro. The \kv{key} returns ??? if such key is not declared.

You can use keys without values in the parameters list too, but with additional care. For example, suppose draft option without parameter. If a user writes \myframe [...] \text{draft, ...}]\text{[text]} then \myframe should behave differently. We have to add DRAFTv=0, in \myframedefaults macro. Moreover, \myframe macro must include preprocessing of \myframedefaults using \replstring which replaces the occurrence of draft by DRAFTv=1.

\optdef\myframe [] #1{...
  \ea\addto\ea\myframedefaults\ea{\the\opt}%
  \replstring\myframedefaults{draft}{DRAFTv=1}%
  \readkv\myframedefaults
...
  \ifnum\kv{DRAFTv}=1 draft mode\else normal mode\fi
...}

\readkv \kv ;

Implementation. The \readkv expands its parameter and does replace-strings in order to remove spaces around equal signs and after commas. Double commas are removed. Then \_kvscan reads the parameters list finished by the double comma and saves values to \_kv:⟨key⟩ macros. The \kv{key} expands the \_kv:⟨key⟩ macro. If this macro isn’t defined then \_kvunknown is processed. You can re-define it if you want.

\readkv{Key-value dictionaries <2020-12-21>} % preloaded in format

2.10 Plain \TeX macros

All macros from plain \TeX are rewritten here. Differences are mentioned in the documentation below.

The \dospecials works like in plain \TeX but does nothing with _. If you need to do the same with this character, you can re-define:

\addto \dospecials{\do\_}

The shortcuts \chardef\@one is not defined in Op\TeX. Use normal numbers instead of such obscurities. The \magstep and \magstephalf are defined with \space, (no \relax), in order to be expandable.

Plain \TeX basic macros and control sequences. \endgraf, \endline. The ^L is not defined in Op\TeX because it is obsolete.
\_\texttt{\_def\}^^M\{\ } % control <return> = control <space>
\_\texttt{\_def\}^^I\{\ } % same for <tab>
\_\texttt{\_def\}lq{`} \_\texttt{\_def\}rq{'}
\_\texttt{\_def\}lbrack{[} \_\texttt{\_def\}rbrack{]}
\_\texttt{\_def\}\texttt{\_endgraf}=\_\texttt{\_par}
\_\texttt{\_let\_endgraf=\_par}
\_\texttt{\_let\_endline=\_cr}
\_\texttt{\_public \_endgraf \_endline ;}

Plain \TeX\ classical \verb|\obeylines| and \verb|\obeyspaces|.

Spaces. \verb|\thinspace|, \verb|\negthinspace|, \verb|\enspace|, \verb|\enskip|, \verb|\quad|, \verb|\qquad|, \verb|\smallskip|, \verb|\medskip|, \verb|\bigskip|, \verb|\nointerlineskip|, \verb|\offinterlineskip|,
\verb|\topglue|, \verb|\vglue|, \verb|\hglue|, \verb|\slash|.

\verb|\protected\_def\_thinspace {\_kern .16667em }|
\verb|\protected\_def\_negthinspace {\_kern-.16667em }|
\verb|\protected\_def\_enspace {\_kern.5em }|
\verb|\protected\_def\_enskip {\_hskip.5em \_relax}|
\verb|\protected\_def\_quad {\_hskip1em \_relax}|
\verb|\protected\_def\_qquad {\_hskip2em \_relax}|
\verb|\protected\_def\_smallskip {\_vskip\_smallskipamount}|
\verb|\protected\_def\_medskip {\_vskip\_medskipamount}|
\verb|\protected\_def\_bigskip {\_vskip\_bigskipamount}|
\verb|\_protected\_def\_nointerlineskip {\_prevdepth=-1000pt }|
\verb|\_protected\_def\_offinterlineskip {\_baselineskip=-1000pt \_lineskip=0pt \_lineskiplimit=\_maxdimen}|
\verb|\_protected\_def\_topglue {\_nointerlineskip\_vglue-\_topskip\_vglue} % for top of page
\verb|\_protected\_def\_vglue {\_afterassignment\_vglA \_skip0=}|
\verb|\_protected\_def\_hglue {\_afterassignment\_hglA \_skip0=}|
\verb|\_protected\_def\_slash {/\_penalty\_exhyphenpenalty} % a `/' that acts like a `-'

Penalties macros: \verb|\break|, \verb|\nobreak|, \verb|\allowbreak|, \verb|\filbreak|, \verb|\goodbreak|, \verb|\eject|, \verb|\supereject|,
\verb|\dosupereject|, \verb|\removelastskip|, \verb|\smallskip|, \verb|\medskip|, \verb|\bigskip|.

56
Boxes. \line, \leftline, \rightline, \centerline, \rlap, \llap, \underbar.

\def \line \{\bbox to \hsize\}
\def \leftline #1\{\line {#1 \hss}\}
\def \centerline #1\{\line {\hss #1 \hss}\}
\def \rlap #1\{\bbox to \zo {#1 \hss}\}
\def \llap #1\{\bbox to \zo {\hss #1}\}
\def \underbar #1\{$\setbox0 = \bbox {#1}\dp0 = \zo \math \underline {\box0}\}$

The \texttt{strutbox} is declared as 10pt size dependent (like in plain \TeX{}), but the macro \texttt{\setbaselineskip} (from \texttt{fonts-opmac.opm}) redefines it.

\newbox \strutbox
\setbox \strutbox = \bbox \{\vrule height 8.5pt depth 3.5pt width 0pt\}
\def \strut \{\relax \ifmmode \copy \strutbox \else \unhcopy \strutbox \fi\}

Alignment. \hidewidth \ialign \multispan.

\def \hidewidth \{\hskip \hideskip\} % for alignment entries that can stick out
\def \ialign \{\everycr = {} \tabskip = \zoskip \halign\}
\newcount \mscount
\def \multispan #1\{\omit \mscount = #1 \relax
\loop \ifnum \mscount > 1 \spanA \repeat
\def \spanA \{\span \omit \advance \mscount by -1 \}

Tabbing macros are omitted because they are obsolete. \texttt{textindent}, \item, \itemitem, \narrower, \raggedright, \tttraggedright, \leavevmode.

\def \hang \{\hangindent \parindent\}
\def \itemitem \{\par \hang \textindent\}
\def \narrower \{\advance \leftskip \parindent \advance \rightskip \parindent\}
\def \raggedright \{\rightskip = 0pt plus 2em \spaceskip = 3333em \xspaceskip = 5em \relax\}
\def \tttraggedright \{\tt \rightskip = 0pt plus 2em \relax\} % for use with \tt only
\def \leavevmode \{\unhbox \voidbox\} % begins a paragraph, if necessary

Few character codes are set for backward compatibility. But old obscurities (from plain \TeX{}) based on \texttt{\mathhexbox} are not supported – an error message and recommendation to directly using the desired character is implemented by the \texttt{\usedirectly} macro. The user can re-define these control sequences of course.
The plain TeX macros \hrulefill, \dotfill, \rightarrowfill, \leftarrowfill, \downbracefill, \upbracefill. The last four are used in non-Unicode variants of \overrightarrow, \overleftarrow, \overbrace and \underbrace macros, see section 2.15.

The last part of plain TeX macros: \magnification, \bye. Note that math macros are defined in the math-macros.opm file (section 2.15).
2.11 Preloaded fonts for text mode

The format in luatex can download only non-Unicode fonts. Latin Modern EC is loaded here. These fonts are totally unusable in LuaTeX when languages with out of ASCII or ISO-8859-1 alphabets are used (for example Czech). We load only a few 8bit fonts here especially for simple testing the format. But, if the user needs to do more serious work, he/she can use \fontfam macro to load a selected font family of Unicode fonts.

We have a dilemma: when the Unicode fonts cannot be preloaded in the format then the basic font set can be loaded by \everyjob. But why to load a set of fonts at the beginning of every job when it is highly likely that the user will load something completely different. Our decision is: there is a basic 8bit font set in the format (for testing purposes only) and the user should load a Unicode font family at beginning of the document.

The fonts selectors \tenrm, \tenbf, \tenit, \tenbi, \tentt are declared as \public here but only for backward compatibility. We don’t use them in the Font Selection System. But the protected versions of these control sequences are used in the Font Selection System.

```
\fontfam
\setfontsize{at10pt}
\resizethefont
```

2.12 Scaling fonts in text mode (low-level macros)

This section describes single part of Font Selection System: resizing fonts to various sizes. This feature is available in both modes: TFM mode (initialized when format starts) and OTF mode (after \fontfam or \initunifonts is used).

2.12.1 The \setfontsize macro

The \setfontsize \{size spec\} saves the information about \langle size spec \rangle. This information is taken into account when a variant selector (for example \rm, \bf, \it, \bi) or \resizethefont is used. The \langle size spec \rangle can be:

- \at\langle dimen \rangle, for example \setfontsize{at12pt}. It gives the desired font size directly.
- \scaled\langle scale factor \rangle, for example \setfontsize{scaled1200}. The font is scaled in respect to its native size (which is typically 10pt). It behaves like \font\ldots scaled\langle number \rangle.
- \mag\langle decimal number \rangle, for example \setfontsize{mag1.2}. The font is scaled in respect to the current size of the fonts given by the previous \setfontsize command.

The initialization value in OpTeX is given by \setfontsize{at10pt}.

The \resizethefont resizes the currently selected font to the size given by previous \setfontsize. For example

```
The 10 pt text is here,
\setfontsize{at12pt} the 10 pt text is here unchanged...
\resizethefont and the 12 pt text is here.
```

The \setfontsize command acts like font modifier. It means that it saves information about fonts but does not change the font actually until variant selector or \resizethefont is used.

The following example demonstrates the \mag format of \setfontsize parameter. It is only a curious example probably not used in practical typography.

```
\def\smaller{\setfontsize{mag.9}\resizethefont}
Text \smaller text \smaller text \smaller text.
```
2.12.2 The \font primitive

If you load a font directly by \font primitive and you want to create a size-dependent selector for such font then you can use \resizethefont:

\font\tencomfortaa=Comfortaa-Regular-T1 at10pt
\def\comfortaa{\tencomfortaa\resizethefont}

\comfortaa The 10 pt text is here
\setfontsize{at12pt}
\comfortaa The 12 pt text is here

The example above uses the 8 bit tfm font. You can use Unicode font too, of course. The \fontfam macro initializes the extended \font primitive features for LuaTeX (see section 2.13.14). If you didn’t use this command, you must initialize these features by the \initunifonts command explicitly, for example:

\initunifonts
\font\tencyklop=[cyklop-regular] at10pt % the font cyklop-regular.otf is loaded
\def\cyklop{\tencyklop\resizethefont}

\cyklop The 10 pt text is here
\setfontsize{at12pt}
\cyklop The 12 pt text is here

2.12.3 The \fontlet declarator

We have another command for scaling: \fontlet which can resize arbitrary font given by its font switch. This font switch was declared by the \font primitive or the \fontdef macro.

\fontlet \( \langle \text{newfont} \rangle \) = \( \langle \text{fontswitch} \rangle \langle \text{sizespec} \rangle \)
example:
\fontlet \bigfont = \_tenbf at15pt

The resulted \bigfont is the same as in the previous example where \fontdef was used. The advantage of \fontdef macro will be more clear when you load font families by \fontfam and you are using more font modifiers declared in such families.

Summary: you can declare font switches:

- by the \font primitive if you know the font file,
- by the \fontlet command if you know the font switch and the size, or
- by the \fontdef command if you know the variant and modifiers.

2.12.4 Optical sizes

There are font families with more font files where almost the same font is implemented in various design sizes: cmr5, cmr6, cmr7, cmr8, cmr9, cmr10, cmr12, cmr17 for example. This feature is called “optical sizes”. OpTeX chooses a font with an optical size closest to desired size specified by the \setfontsize, when at\langle \text{dimen} \rangle or mag\langle \text{coefficient} \rangle is used. When scaled\langle \text{scale factor} \rangle is used then optical size is chosen using the value of the \defaultoptsize register and such font is scaled by the specified \langle \text{scale factor} \rangle.

There is \defaultoptsize=10pt by default.

Font collections with optical sizes must be registered by the \_regtfm for tfm files or \_regoptsizes for Unicode fonts. OpTeX registers 8bit Latin Modern fonts in the format and OTF Latin Modern fonts in the f-lmfonts.opm file. See also section 2.13.12.

2.12.5 Implementation of resizing

Only “resizing” macros are implemented here. Other aspects of Font Selection System and their implementation are described in section 2.13.15.

\setfontsize \{\langle \text{sizespec} \rangle \} saves the \langle \text{sizespec} \rangle to the \_sizespec macro. The \_optsize value is calculated from the \langle \text{sizespec} \rangle. If the \langle \text{sizespec} \rangle is in the mag\langle \text{number} \rangle format then the contents of the \_sizespec macro is re-calculated to the at\langle \text{dimen} \rangle format using previous \_optsize value.
\newdimen \optsize \optsize=10pt
\newdimen \defaultoptsize \defaultoptsize=10pt
\newdimen \lastmagsize
\def \setfontsize #1{%
  \edef \sizespec{#1}%
  \ea \setoptsize \sizespec \relax
  \releading
}
\def \setoptsize {
  \isnextchar a{ \setoptsizeA}{\isnextchar m{ \setoptsizeC}{ \setoptsizeB}}}
\def \setoptsizeA at#1 \relax{\optsize=#1 \relax \lastmagsize=\optsize} % at<dimen>
\def \setoptsizeB scaled#1 \relax{\optsize=\defaultoptsize \relax} % scaled<scalenum>
\def \setoptsizeC mag#1 \relax{
  \ifdim \lastmagsize > \zo \optsize=\lastmagsize \else \optsize=\pdffontsize \font \fi
  \optsize=#1 \optsize
  \lastmagsize=\optsize
  \edef \sizespec{at \the \optsize}%
}
\public \setfontsize \defaultoptsize ;
\fontlet ⟨font switch A⟩ ⟨font switch B⟩ ⟨size spec⟩
does
\font ⟨font switch A⟩ = ⟨fontname⟩ ⟨sizespec⟩
The ⟨fontname⟩ is extracted using the primitive command \fontname ⟨font switch B⟩.
The \fontname is in the form as shown below in the code where \fontname is used.
\newcurrfontsize ⟨size spec⟩ sets current font size to the ⟨size spec⟩ It is implemented by \fontlet.
The font switch of the current font is extracted by \the \font. We must re-create the control sequence \the \font because its original meaning is set to “inaccessible” by \font when the font primitive is started.
\resizethefont is implemented by \newcurrfontsize using data from the \sizespec macro.
The \regtfm ⟨font id⟩ ⟨optical size data⟩ saves the ⟨optical size data⟩ concerned to ⟨font id⟩. The ⟨optical size data⟩ is in the form as shown below in the code where \regtfm is used.
The \wichtfm ⟨fontname⟩ expands to the ⟨fontname⟩ or to the corrected ⟨fontname⟩ read from the ⟨optical size data⟩. It is used in the \rfontskipat macro and it is used in \fontlet macro. It means that each ⟨fontname⟩ generated by the \fontname primitive in the \fontlet macro is processed by the \wichtfm. The real ⟨fontname⟩ or corrected ⟨fontname⟩ (depending on the optical data does not exist or exist) is the output of the expansion before \font primitive takes this output as its parameter.
The implementation detail: The \begin{verbatim}
\def \regtfm #1 0 #2 *{
  \def \csname _#1:reg\endcsname{\#2 16380} %
  \def \tmpa{#1} \reversetfm #2 *
}
\def \reversetfm #1 #2 {%
  we need this data for \setmathfamily
  \let \csname _#1:reg\endcsname\endname
}
\protected \def \resizethefont{\newcurrfontsize \sizespec}
\public \newcurrfontsize \resizethefont ;
\end{verbatim}
The \begin{verbatim}
\def \regtfm #1 \#2 \#3 +*{\def \csname _#1:reg\endcsname(#2 \#3 16380) \relax} %
\def \tmpa(#1) \reversetfm #1 \#3 * %
}
\def \reversetfm #1 \#2 \#3 {%
  we need this data for \setmathfamily
  \let \csname _#1:reg\endcsname\endname
}
\protected \def \resizethefont{\newcurrfontsize \sizespec}
\public \newcurrfontsize \resizethefont ;
\end{verbatim}

62
Optical sizes data for preloaded 8bit Latin Modern fonts:

<table>
<thead>
<tr>
<th>Font</th>
<th>Size (pt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lmr</td>
<td>0</td>
</tr>
<tr>
<td>lmr5</td>
<td>5.5</td>
</tr>
<tr>
<td>lmr6</td>
<td>6.5</td>
</tr>
<tr>
<td>lmr7</td>
<td>7.5</td>
</tr>
<tr>
<td>lmr8</td>
<td>8.5</td>
</tr>
<tr>
<td>lmr9</td>
<td>9.5</td>
</tr>
<tr>
<td>lmbx</td>
<td>0</td>
</tr>
<tr>
<td>lmbx5</td>
<td>5.5</td>
</tr>
<tr>
<td>lmbx6</td>
<td>6.5</td>
</tr>
<tr>
<td>lmbx7</td>
<td>7.5</td>
</tr>
<tr>
<td>lmbx8</td>
<td>8.5</td>
</tr>
<tr>
<td>lmbx9</td>
<td>9.5</td>
</tr>
</tbody>
</table>

2.13 The Font Selection System

The basic principles of the Font Selection System used in OpTeX was documented in the section 1.3.1.

2.13.1 Terminology

We distinguish between

- **font switchers**, they are declared by the \font primitive or by \fontlet or \fontdef macros, they select given font.

- **variant selectors**, there are four basic variant selectors \rm, \bf, \it, \bi, there is a special selector \currvar. More variant selectors can be declared by the \famvardef macro. They select the font depending on the given variant and on the font context (i.e. on current family and on more features given by font modifiers). In addition, OpTeX defines \tt as variant selector independent of chosen font family. It selects typewriter-like font.

- **font modifiers** are declared in a family (\cond, \caps) or are “built-in” (\setfontsize{⟨size spec⟩}, \setff{⟨features⟩}). They do appropriate change in the font context but do not select the font.

- **family selectors** (for example \Termes, \LMfonts), they are declared typically in the font family files. They enable to switch between font families, they do appropriate change in the font context but do not select the font.

These commands set their values locally. When the \TeX group is left then the selected font and the font context are returned back to the values used when the group was opened. They have the following features:

The **font context** is a set of macro values that will affect the selection of real font when the variant selector is processed. It includes the value of current family, current font size, and more values stored by font modifiers.

The **family context** is the current family value stored in the font context. The variant selectors declared by \famvardef and font modifiers declared by \moddef are dependent on the family context. They can have the same names but different behavior in different families.

The fonts registered in OpTeX have their macros in the font family files, each family is declared in one font family file with the name f-\famname.omp. All families are collected in fams-ini.omp and users can give more declarations in the file fams-local.omp.

2.13.2 Font families, selecting fonts

The \fontfam[⟨Font Family⟩] opens the relevant font family file where the ⟨Font Family⟩ is declared. The family selector is defined here by rules described in the section 2.13.11. Font modifiers and variant selectors may be declared here. The loaded family is set as current and \rm variant selector is processed.

The available declared font modifiers and declared variant selectors are listed in the log file when the font family is load. Or you can print \fontfam[catalog] to show available font modifiers and variant selectors.
The font modifiers can be independent, like \cond and \light. They can be arbitrarily combined (in arbitrary order) and if the font family disposes of all such sub-variants then the desired font is selected (after variant selector is used). On the other hand, there are font modifiers that negates the previous font modifier, for example: \cond, \extend. You can reset all modifiers to their initial value by the \resetmod command.

You can open more font families by more \fontfam commands. Then the general method to selecting the individual font is:

\begin{verbatim}
(family selector) \langle font modifiers \rangle \langle variant selector \rangle
\end{verbatim}

For example:

\begin{verbatim}
\fontfam [Heros] \% Heros family is active here, default \rm variant.
\fontfam [Termes] \% Termes family is active here, default \rm variant.
{\Heros \caps \cond \it The caps+condensed italics in Heros family is here.}
The Termes roman is here.
\end{verbatim}

There is one special command \currvar which acts as a variant selector. It keeps the current variant and the font of such variant is reloaded with respect to the current font context by the previously given family selector and font modifiers.

You can use the \setfontsize \{(sizespec)\} command in the same sense as other font modifiers. It saves information about font size to the font context. See section 2.12. Example:

\begin{verbatim}
\rm default size \setfontsize{(at14pt)}\rm here is 14pt size \it italic is in 14pt size too \bf bold too.
\end{verbatim}

A much more comfortable way to resize fonts is using OPmac-like commands \typosize and \typoscale. These commands prepare the right sizes for math fonts too and they re-calculate many internal parameters like \baselineskip. See section 2.17 for more information.

2.13.3 Math Fonts

Most font families are connected with a preferred Unicode-math font. This Unicode-math is activated when the font family is loaded. If you don’t prefer this and you are satisfied with 8bit math CM+AMS fonts preloaded in the OpTeX format then you can use command \noloadmath before you load a first font family.

If you want to use your specially selected Unicode-math font then use \loadmath \{(font_file)\} or \loadmath \{(font_name)\} before first \fontfam is used.

2.13.4 Declaring font commands

Font commands can be font switches, variant selectors, font modifiers, family selectors and defined font macros doing something with fonts.

- Font switches can be declared by \font primitive (see section 2.12.2) or by \fontlet command (see section 2.12.3) or by \fontdef command (see sections 2.13.5). When the font switches are used then they select the given font independently of the current font context. They can be used in \output routine (for example) because we need to set fixed fonts in headers and footers.

- Variant selectors are \rm, \bf, \it, \bi, \tt and \currvar. More variant selectors can be declared by \famvardef command. They select a font dependent on the current font context, see section 2.13.6. The \tt selector is documented in section 2.13.7.

- Font modifiers are “built-in” or declared by \moddef command. They do modifications in the font context but don’t select any font.
  - “built-in” font modifiers are \setfontsize (see section 2.12), \setff (see section 2.13.9), \setfontcolor, \setletterspace and \setwordspace (see section 2.13.10). They are independent of font family.

- Font modifiers declared by \moddef depend on the font family and they are typically declared in font family files, see section 2.13.11.

- Family selectors set the given font family as current and re-set data used by the family-dependent font modifiers to initial values and to the currently used modifiers. They are declared in font family files by \_famdecl macro, see section 2.13.11.

- Font macros can be defined arbitrarily by \def primitive by users. See an example in section 2.13.8.
All declaration commands mentioned here: \font, \fontlet, \fontdef, \famvardef, \moddef, \_famdec1 and \def make local assignment.

2.13.5 The \fontdef declarator in detail

You can declare \fontdef by the \fontdef command.

\fontdef\(\text{font-switch}\) \{\(\text{family selector}\) \{\(\text{font modifiers}\) \(\text{variant selector}\)\}\}

where \(\text{family selector}\) and \(\text{font modifiers}\) are optional and \(\text{variant selector}\) is mandatory. The resulting \fontdef declared by \\fontdef is “fixed font switch” independent of the font context. More exactly, it is a fixed font switch when it is used. But it can depend on the current font modifiers and font family and given font modifiers when it is declared.

The \fontdef does the following steps. It pushes the current font context to a stack, it does modifications of the font context by given \(\text{family selector}\) and/or \(\text{font modifiers}\) and it finds the real font by \(\text{variant selector}\). This font is not selected but it is assigned to the declared \fontdef (like \font primitive does it). Finally, \fontdef pops the font context stack, so the current font context is the same as it was before \fontdef is used.

2.13.6 The \famvardef declarator

You can declare a new variant selector by the \famvardef macro. This macro has similar syntax as \fontdef:

\famvardef\(\text{new variant selector}\) \{\(\text{family selector}\) \{\(\text{font modifiers}\) \(\text{variant selector}\)\}\}

where \(\text{family selector}\) and \(\text{font modifiers}\) are optional and \(\text{variant selector}\) is mandatory. The \(\text{new variant selector}\) declared by \famvardef should be used in the same sense as \rm, \bf etc. It can be used as the final command in next \fontdef or \famvardef declarators too. When the \(\text{new variant selector}\) is used in the normal text then it does the following steps: pushes current font context to a stack, modifies font context by declared \(\text{family selector}\) and/or \(\text{font modifiers}\), runs following \(\text{variant selector}\). This last one selects a real font. Then pops the font context stack. The new font is selected but the font context has its original values. This is main difference between \famvardef\foo{...} and \def\foo{...}.

Moreover, the \famvardef creates the \(\text{new variant selector}\) family dependent. When the selector is used in another family context than it is defined then a warning is printed on the terminal “\text{var selector}” is undeclared in the current family” and nothing happens. But you can declare the same variant selector by \famvardef macro in the context of a new family. Then the same command may do different work depending on the current font family.

Suppose that the selected font family provides the font modifier \medium for mediate weight of fonts. Then you can declare:

\famvardef \mf \{\medium \rm\}
\famvardef \mi \{\medium \it\}

Now, you can use six independent variant selectors \rm, \bf, \it, \bi, \mf and \mi in the selected font family.

A \(\text{family selector}\) can be written before \(\text{font modifiers}\) in the \famvardef parameter. Then the \(\text{new variant selector}\) is declared in the current family but it can use fonts from another family represented by the \(\text{family selector}\).

When you are mixing fonts from more families then you probably run into a problem with incompatible ex-heights. This problem can be solved using \setfontsize and \famvardef macros:

\fontfam[\text{Heros}] \fontfam[\text{Termes}]
\defexhcorr{\setfontsize{\text{mag.88}}}
\famvardef[\text{rmsans}][\text{\text{Heros}}][\text{exhcorr}][\text{\rm}]
\famvardef[\text{itsans}][\text{\text{Heros}}][\text{exhcorr}][\text{\it}]

Compare ex-height of \text{Termes} \text{\rmsans} with \text{Heros} \text{\rm} and \text{Termes}.

The variant selectors (declared by \famvardef) or font modifiers (declared by \moddef) are (typically) control sequences in user name space (\mf, \caps). They are most often declared in font family
files and they are loaded by \fontfam. A conflict with such names in user namespace can be here. For example: if \mf is defined by a user and then \fontfam[Roboto] is used then \famvardef\mf is performed for Roboto family and the original meaning of \mf is lost. But OpTeX prints warning about it. There are two cases:

\begin{verbatim}
def\mf{Metafont}
  \fontfam[Roboto] \% warning: "The mf is redefined by \famvardef" is printed or
\fontfam[Roboto]
def\mf{Metafont} \% \mf variant selector redefined by user, we suppose that \mf \% is used only in the meaning of "Metafont" in the document.
\end{verbatim}

2.13.7 The \tt variant selector

\tt is an additional special variant selector which is defined as “select typewriter font independently of the current font family”. By default, the typewriter font-face from LatinModern font family is used.

The \tt variant selector is used in OpTeX internal macros \_ttfont (verbatim texts) and \_urlfont (printing URL’s).

You can redefine the behavior of \tt by \famvardef. For example:

\begin{verbatim}
def\fontfam[Cursor]
def\fontfam[Heros]
def\fontfam[Termes]
famvardef\tt\{\Cursor\setff{-liga;-tlig}\rm}
\end{verbatim}

Test in Termes: \tt text. \{\Heros\rm Test in Heros: \tt text\}. Test in URL \url{http://something.org}.

You can see that \tt stay family independent. This is a special feature only for \tt selector. New definition is used in \_ttfont and \_urlfont too. It is recommended to use \setff{-liga;-tlig} to suppress the ligatures in typewriter fonts.

If Unicode math font is loaded then the \tt macro selects typewriter font-face in math mode too. This face is selected from used Unicode math font and it is independent of \famvardef\tt declaration.

2.13.8 Font commands defined by \def

Such font commands can be used as fonts selectors for titles, footnotes, citations, etc. Users can define them.

The following example shows how to define a “title-font selector”. Titles are not only bigger but they are typically in the bold variant. When a user puts {\it...} into the title text then he/she expects bold italic here, no normal italic. You can remember the great song by John Lennon “Let It Be” and define:

\begin{verbatim}
def\titlefont{\setfontsize{at14pt}\bf \let\it\bi}
\end{verbatim}

... \{\titlefont Title in bold 14pt font and \{\it bold 14pt italics\} too\}

OpTeX defines similar internal commands \_titfont, \_chapfont, \_secfont and \_seccfont, see section 2.26. The commands \typosize and \boldify are used in these macros. They set the math fonts to given size too and they are defined in section 2.17.

2.13.9 Modifying font features

Each OTF font provides “font features”. You can list these font features by otfinfo -f font.otf. For example, LinLibertine fonts provide \frac font feature. If it is active then fractions like 1/2 are printed in a special form.

The font features are part of the font context data. The macro \setff {⟨feature⟩} acts like family independent font modifier and prepares a new ⟨feature⟩. You must use a variant selector in order to reinitialize the font with the new font feature. For example \setff{+frac}\rm or \setff{+frac}\currvar. You can declare a new variant selector too:

\begin{verbatim}
def\fontfam[LinLibertine]
  \famvardef \fraclig {\setff{+frac}\currvar}
\end{verbatim}

Compare 1/2 or 1/10 \fraclig to 1/2 or 1/10.
If the used font does not support the given font feature then the font is reloaded without warning nor error, silently. The font feature is not activated.

The \texttt{onum} font feature (old-style digits) is connected to \texttt{\caps} macro for Caps+SmallCaps variant in Op\TeX\ font family files. So you need not create a new modifier, just use \texttt{\caps\currvar 012345}.

### 2.13.10 Special font modifiers

Despite the font modifiers declared in the font family file (and dependent on the font family), we have following font modifiers (independent of font family):

\begin{align*}
&\texttt{\setfontsize{\langle sizespec\rangle}} \quad \text{\% sets the font size} \\
&\texttt{\setff{\langle font\ feature\rangle}} \quad \text{\% adds the font feature} \\
&\texttt{\setfontcolor{\langle color\rangle}} \quad \text{\% sets font color} \\
&\texttt{\setletterspace{\langle number\rangle}} \quad \text{\% sets letter spacing} \\
&\texttt{\setwordspace{\langle scaling\rangle}} \quad \text{\% modifies word spacing}
\end{align*}

The \texttt{\setfontsize} command is described in the section 2.12. The \texttt{\setff} command was described in previous subsection.

\texttt{\setfontcolor{\langle color\rangle}} specifies the color and the opacity of the text. The \texttt{\langle color\rangle} parameter should be in the hexadecimal format of four bytes \texttt{\langle red\rangle\langle green\rangle\langle blue\rangle\langle opacity\rangle}, for example \texttt{FF0080FF} means full red, zero green, half blue and full opacity. You can use names \texttt{red}, \texttt{green}, \texttt{blue}, \texttt{yellow}, \texttt{cyan}, \texttt{magenta}, \texttt{white}, \texttt{grey}, \texttt{lgrey} (without the backslash) instead of the hexadecimal specification. The empty parameter \texttt{\langle color\rangle} means default black color.

These colors of fonts are implemented using Lua\TeX\ internal font feature. This is different approach than using colors in section 2.20.

\texttt{\setletterspace{\langle number\rangle}} specifies the letter spacing of the font. The \texttt{\langle number\rangle} is a decimal number without unit. The unit is supposed as 1/100 of the font size. I.e. 2.5 means 0.25 pt when the font is at 10 pt size. The empty parameter \texttt{\langle number\rangle} means no letter spacing which is the default.

\texttt{\setwordspace{\langle scaling\rangle}} scales the default interword space (defined in the font) and its stretching and shrinking parameters by given \texttt{\langle scaling\rangle} factor. For example \texttt{\setwordspace{2.5}} multiplies interword space by 2.5. \texttt{\setwordspace} can use different multiplication factors if its parameter is in the format \texttt{{/\langle default\rangle/\langle stretching\rangle/\langle shrinking\rangle}}. For example, \texttt{\setwordspace{/1/2.5/1}} enlarges only stretching 2.5 times.

You can use \texttt{\setff} with other font features provided by Lua\TeX\ and \textit{luaotfload} package (see documentation of \textit{luaotfload} package for more information):

\begin{align*}
&\texttt{\setff{embolden=1.5}} \texttt{\rm} \quad \text{\% font is bolder because outline has nonzero width} \\
&\texttt{\setff{slant=0.2}} \texttt{\rm} \quad \text{\% font is slanted by a linear transformation} \\
&\texttt{\setff{extend=1.2}} \texttt{\rm} \quad \text{\% font is extended by a linear transformation.} \\
&\texttt{\setff{colr=yes}} \texttt{\rm} \quad \text{\% if the font includes colored characters, use colors} \\
&\texttt{\setff{upper}} \texttt{\rm} \quad \text{\% to uppercase (lower=lowercase) conversion at font level} \\
&\texttt{\setff{fallback=name}} \texttt{\rm} \quad \text{\% use fonts from a list given by name if missing chars}
\end{align*}

Use font transformations \texttt{embolden}, \texttt{slant}, \texttt{extend} and \texttt{\setletterspace}, \texttt{\setwordspace} with care. The best setting of these values is the default setting in every font, of course. If you really need to set a different letter spacing then it is strongly recommended to add \texttt{\setff{-liga}} to disable ligatures. And setting a positive letter spacing probably needs to scale interword spacing too.

All mentioned font modifiers (except for \texttt{\setfontsize}) work only with Unicode fonts loaded by \texttt{\fontfam}.

### 2.13.11 How to create the font family file

The font family file declares the font family for selecting fonts from this family at the arbitrary size and with various shapes. Unicode fonts (OTF) are preferred. The following example declares the Heros family:

\begin{verbatim}
\_famdecl [Heros] \Heros \{TeX Gyre Heros fonts based on Helvetica\} \\
\{\caps\cond\} \{\bf \it \bi\} \{\FiraMath\} \\
\{\texgyreheros-regular\} \\
\{\_def\_fontnamegen{\texgyreheros\_condV\_\currV:\_capsV\_\fontfeatures}\} \\
\_wlog{\_detokenize{%}
\end{verbatim}
If you want to write such a font family file, you need to keep the following rules.

- Use the \famdecl command first. It has the following syntax:

\famdecl \[\langle Name of family\rangle \langle Familyselector\rangle \{\langle comments\rangle\}\{\langle modifiers\rangle\}\{\langle variant selectors\rangle\}\{\langle comments about math fonts\rangle\}\{\langle font-for-testing\rangle\}\{\langle def\_fontnamegen\{\langle font name or font file name generated\rangle\}\}\)

This writes information about font family at the terminal and prevents loading such file twice. Moreover, it probes existence of \{\langle font-for-testing\rangle\} in your system. If it doesn’t exist, the file loading is skipped with a warning on the terminal. The \_famvardef macro returns false in this case. The \_fontnamegen macro must be defined in the last parameter of the \famdecl. More about it is documented below.

- You can use \_wlog{\_detokenize{...} to write additional information into a log file.
- You can declare optical sizes using \regoptsizes if there are more font files with different optical sizes (like in Latin Modern). See flmfonts.ope file for more information about this special feature.
- Declare font modifiers using \moddef if they are present. The \resetmod must be declared in each font family.
- Check if all your declared modifiers do not produce any space in horizontal mode. For example: X\caps Y, the letters XY must be printed without any space.
- Optionally, declare new variants by the \famvardef macro.
- Run \_initfontfamily to start the family (it is mandatory).
- If math font should be loaded, use \_loadmath{\langle math font\rangle}.

The \_fontnamegen macro (declared in the last parameter of the \famdecl) must expand (at the expand processor level only) to a file name of the loaded font (or to its font name) and to optional font features appended. The Font Selection System uses this macro at the primitive level in the following sense:

\font \{\langle font-switch\rangle \{\_fontnamegen\} \{\_sizespec\}

Note that the extended \font syntax \font\{\langle font-switch\rangle \{\langle font name\rangle:\langle font features\rangle\} \{\langle size spec\rangle\} or \font\{\langle font-switch\rangle \{\langle font file name\rangle:\langle font features\rangle\} \{\langle size spec\rangle\} is expected here.
Example 1

Assume an abstract font family with fonts \texttt{xx-Regular.otf}, \texttt{xx-Bold.otf}, \texttt{xx-Italic.otf} and \texttt{xx-BoldItalic.otf}. Then you can declare the \texttt{\resetmod} (for initializing the family) by:

\begin{verbatim}
\_moddef\resetmod{\_fvars Regular Bold Italic BoldItalic }
\end{verbatim}

and define the \texttt{\_fonnamegen} in the last parameter of the \texttt{\_famdecl} by:

\begin{verbatim}
\_famdecl ...
  {\def\_fonnamegen{[xx-\_currV]}}
\end{verbatim}

The following auxiliary macros are used here:

- \texttt{\moddef} declares the family dependent modifier. The \texttt{\resetmod} saves initial values for the family.
- \texttt{\_fvars} saves four names to the memory, they are used by the \texttt{\_currV} macro.
- \texttt{\_currV} expands to one of the four names dependent on \texttt{\rm} or \texttt{\bf} or \texttt{\it} or \texttt{\bi} variant is required.

Assume that the user needs \texttt{\it} variant in this family. Then the \texttt{\_fonnamegen} macro expands to \texttt{[xx-\_currV]} and it expands to \texttt{[xx-Italic]}. The Font Selection System uses \texttt{\font{[xx-Italic]}}.

This command loads the \texttt{xx-Italic.otf} font file.

See more advanced examples are in \texttt{f-⟨family⟩.opm} files.

Example 2

The \texttt{f-heros.opm} is listed here. Look at it. When Heros family is selected and \texttt{\bf} is asked then \texttt{\font{[texgyreheros-bold]:+tlig;}} at10pt is processed.

You can use any expandable macros or expandable primitives in the \texttt{\_fonnamegen} macro. The simple macros in our example with names \texttt{\_fsetV} are preferred. They expand typically to their content. The macro \texttt{\_fsetV (word)=⟨content⟩ (terminated by a space)} is equivalent to \texttt{\def\_⟨word⟩V{⟨content⟩}} and you can use it in font modifiers. You can use the \texttt{\_fsetV} macro in more general form:

\begin{verbatim}
\_fsetV ⟨word-a⟩=⟨value-a⟩,⟨word-b⟩=⟨value-b⟩ ...etc. terminated by a space
\end{verbatim}

with obvious result \texttt{\def\_⟨word-a⟩V {⟨value-a⟩}\def\_⟨word-b⟩V {⟨value-b⟩} etc.}

Example 3

If both font modifiers \texttt{\caps}, \texttt{\cond} were applied in Heros family, then \texttt{\def\_capsV{+smcp;\_ffonum;}} and \texttt{\def\_condV{cn}} were processed by these font modifiers. If a user needs the \texttt{\bf} variant at 11pt now then the

\begin{verbatim}
\font{[texgyreheroscn-bold]:+smcp;+onum;+pnum;+tlig;} at11pt
\end{verbatim}

is processed. We assume that a font file \texttt{texgyreheroscn-bold.otf} is present in your \TeX{} system.

The \texttt{\onlyif} macro

has the syntax \texttt{\onlyif ⟨word⟩=⟨value-a⟩,⟨value-b⟩,...⟨value-n⟩: ⟨⟨what⟩⟩}. It can be used inside \texttt{\moddef} as simple IF statement: the \texttt{⟨what⟩} is processed only if \texttt{⟨word⟩} has \texttt{⟨value-a⟩} or \texttt{⟨value-b⟩} ... or \texttt{⟨value-n⟩}. See \texttt{f-roboto.opm} for examples of usage of many \texttt{\onlyif}’s.

Recommendation: use the \texttt{\fontfeatures} macro at the end of the \texttt{\_fonnamegen} macro in order to the \texttt{\setff}, \texttt{\setfontcolor}, \texttt{\setletterspace} macros can work.

The \texttt{\moddef} macro

has the syntax \texttt{\moddef⟨modifier⟩{⟨what to do⟩}. It does more things than simple \texttt{\def}:}

- The modifier macros are defined as \texttt{\protected}.
- The modifier macros are defined as family-dependent.
- If the declared control sequence is defined already (and it is not a font modifier) then it is re-defined with a warning.

The \texttt{\famvardef} macro has the same features.

The \texttt{\_/Familyselector} is defined by the \texttt{\_famdecl} macro as:

\begin{verbatim}
\protected\def\_/Familyselector {%
  \_def\_currfamily {⟨Familyselector⟩}%;
  \_def\_fonnamegen {⟨Familyselector⟩}%;
  \resetmod
  ⟨run all family-dependent font modifiers used before Familyselector without warnings⟩
\end{verbatim}
The \_initfontfamily must be run after modifier’s declaration. It runs the \Familyselector and it runs \_rm, so the first font from the new family is loaded and it is ready to use it.

Name conventions
Create font modifiers, new variants, and the \Familyselector only as public, i.e. in user namespace without \_ prefix. We assume that if a user re-defines them then he/she needs not them, so we have no problems. If the user’s definition was done before loading the font family file then it is re-defined and Op\TeX{} warns about it. See the end of section 2.13.4.

The name of \Familyselector should begin with an uppercase letter.

Please, look at Op\TeX{} font catalogue before you will create your font family file and use the same names for analogical font modifiers (like \cond, \caps, \sans, \mono etc.) and for extra variant selectors (like \lf, \li, \kf, \ki etc. used in Roboto font family).

If you are using the same font modifier names to analogical font shapes then such modifiers are kept when the family is changed. For example:

\begin{verbatim}
\fontfam [Termes] \fontfam[Heros]
\caps\cond\it Caps+Cond italic in Heros \Termes\currvar Caps italic in Termes.
\end{verbatim}

The family selector first resets all modifiers data by \resetmod and then it tries to run all currently used family-dependent modifiers before the family switching (without warnings if such modifier is unavailable in the new family). In this example, \Termes does \resetmod followed by \caps\cond. The \caps is applied and \cond is silently ignored in Termes family.

If you need to declare your private modifier (because it is used in other modifiers or macros, for example), use the name \_wordM. You can be sure that such a name does not influence the private namespace used by Op\TeX{}.

Additional notes
See the font family file f-libertine-s.opm which is another example where no font files but font names are used.

See the font family file f-lmfonts.opm or f-poltawski.opm where you can find the example of the optical sizes declaration including documentation about it.

If you need to create a font family file with a non-Unicode font, you can do it. The \fontnamegen must expand to the name of TFM file in this case. But we don’t prefer such font family files, because they are usable only with languages with alphabet subset to ISO-8859-1 (Unicodes are equal to letter’s codes of such alphabets), but middle or east Europe use languages where such a condition is not true.

2.13.12 How to write the font family file with optical sizes
You can use \optname macro when \fontnamegen in expanded. This macro is fully expandable and its input is \intemplate and its output is a part of the font file name \sizetemplate with respect to given optical size.

You can declare a collection of \sizetemplates for one given \intemplate by the \regoptsizes macro. The syntax is shown for one real case:

\begin{verbatim}
\regoptsizes lmr.r lmroman?-regular
5 <5.5 6 <6.5 7 <7.5 8 <8.5 9 <9.5 10 <11.1 12 <15 17 <*
\end{verbatim}

In general:

\begin{verbatim}
\regoptsizes \intemplate \generaloutputtemplate \resizingdata
\end{verbatim}

Suppose our example above. Then \optname{lr.m} expands to lmroman?-regular where the question mark is substituted by a number depending on current \optsize. If the \optsize lies between two boundary values (they are prefixed by < character) then the number written between them is used. For example if 11.1 < \optsize <= 15 then 12 is substituted instead question mark. The \resizingdata virtually begins with zero <0, but it is not explicitly written. The right part of \resizingdata must be terminated by <*> which means "less than infinity".

If \optname gets an argument which is not registered \intemplate then it expands to \failedoptname which typically ends with an error message about missing font. You can redefine \failedoptname macro to some existing font if you find it useful.

We are using a special macro \_Lmregfont in f-lmfonts.opm. It sets the file names to lowercase and enables us to use shortcuts instead of real \resizingdata. There are shortcuts \regoptFS, \regoptT,
The collection of \langle internal-templates \rangle are declared, each of them covers a collection of real file names.

The \_optfontalias \{\langle new-template \rangle \} \{\langle internal-template \rangle \} declares \langle new-template \rangle with the same meaning as previously declared \langle internal-template \rangle.

The \_optname macro can be used even if no optical sizes are provided by a font family. Suppose that font file names are much more chaotic (because artists are very creative people), so you need to declare more systematic \langle internal-templates \rangle and do an alias from each \langle internal-template \rangle to \langle real-font-name \rangle. For example, you can do it as follows:

\def\fontalias #1 #2 {\_regoptsizes #1 ?#2 {} <* }
% alias name real font name
\fontalias crea-a-regular {Creative Font}
\fontalias crea-a-bold {Creative FontBold}
\fontalias crea-a-italic {Creative olique}
\fontalias crea-a-bolditalic {Creative Bold plus italic}
\fontalias crea-b-regular {Creative Regular subfam}
\fontalias crea-b-bold {Creative subfam bold}
\fontalias crea-b-italic {Creative-subfam Oblique}
\fontalias crea-b-bolditalic {Creative Bold subfam Oblique}

Another example of a font family with optical sizes is Antykwa Półtawskiego. The optical sizes feature is deactivated by default and it is switched on by \_osize font modifier:

\_initfontfamily % new font family must be initialized

2.13.13 How to register the font family in the Font Selection System

Once you have prepared a font family file with the name f-\langle famname \rangle .opm and \TeX{} can see it in your filesystem then you can type \fontfam \[ \langle famname \rangle \] and the file is read, so the information about the font family is loaded. The name \langle famname \rangle must be lowercase and without spaces in the file name f-\langle famname \rangle .opm. On the other hand, the \fontfam command is more tolerant: you can write uppercase letters and spaces here. The spaces are ignored and uppercase letters are converted to lowercase. For example \fontfam [LM Fonts] is equivalent to \fontfam [LMfonts] and both commands load the file f-lmfonts.opm.

You can use your font file in sense of the previous paragraph without registering it. But problem is that such families are not listed when \fontfam[?] is used and it is not included in the font catalog when
The list of families taken in the catalog and listed on the terminal is declared in two files: \texttt{fams-ini.opm} and \texttt{fams-local.opm}. The second file is optional. Users can create it and write to it the information about user-defined families using the same syntax as in existed file \texttt{fams-ini.opm}.

The information from the user’s \texttt{fams-local.opm} file has precedence. For example \texttt{fams-ini.opm} declares aliases Times\rightarrow Termes etc. If you have the original Times purchased from Adobe then you can register your declaration of Adobe’s Times family in \texttt{fams-local.opm}. When a user writes \texttt{\fontfam[Times]} then the original Times (not Termes) is used.

The \texttt{fams-ini.opm} and \texttt{fams-local.opm} files can use the macros \_faminfo, \_famalias and \_famtext. See the example from \texttt{fams-ini.tex}:

\begin{verbatim}
\fontfam[catalog] is printed. The list of families taken in the catalog and listed on the terminal is declared in two files: \texttt{fams-ini.opm} and \texttt{fams-local.opm}. The second file is optional. Users can create it and write to it the information about user-defined families using the same syntax as in existed file \texttt{fams-ini.opm}.

The information from the user's \texttt{fams-local.opm} file has precedence. For example \texttt{fams-ini.opm} declares aliases Times\rightarrow Termes etc. If you have the original Times purchased from Adobe then you can register your declaration of Adobe’s Times family in \texttt{fams-local.opm}. When a user writes \texttt{\fontfam[Times]} then the original Times (not Termes) is used.

The \texttt{fams-ini.opm} and \texttt{fams-local.opm} files can use the macros \_faminfo, \_famalias and \_famtext. See the example from \texttt{fams-ini.tex}:

\begin{verbatim}
\% Version <2020-02-28>. Loaded in format and secondly on demand by \fontfam[catalog]
\_famtext {Special name for printing a catalog :}
\_faminfo [Catalogue] {Catalogue of all registered font families} {fonts-catalog} {} 
\_famalias [Catalog]
\_famtext {Computer Modern like family:}
\_faminfo [Latin Modern] {TeX Gyre fonts based on Computer Modern} {f-lmfonts}
\{ -\caps: \rm\bf\it\bi \caps: \rm\it \ttlight,\ttcond,\dunhill: \rm\bf\it\bi \upital: \rm\}
\_famalias [LMfonts] \_famalias [Latin Modern Fonts] \_famalias [lm]
\_famtext {TeX Gyre fonts based on Adobe 35:}
\_faminfo [Termes] {TeX Gyre Termes fonts based on Times} {f-termes}
\{ -\caps: \rm\bf\it\bi \caps,\caps\cond,\caps\cond: \rm\bf\it\bi \}
\_famalias [Times]
\_famtext {TeX Gyre Heros fonts based on Helvetica} {f-heros}
\{ -\caps,\cond,\caps\cond: \rm\bf\it\bi \}
\_famalias [Helvetica]
\end{verbatim}

... etc.

The \_faminfo command has the syntax:

\_faminfo [\{Family Name\}] {\{comments\}} {\{file-name\}}
  {\{mod-plus-vars\}}

The \texttt{(mod-plus-vars)} data is used only when printing the catalog. It consists of one or more pairs \texttt{(mods) : {\{vars\}}}. For each pair: each modifier (separated by comma) is applied to each variant selector in \texttt{\{vars\}} and prepared samples are printed. The - character means no modifiers should be applied.

The \_famalias declares an alias to the last declared family.

The \_famtext writes a line to the terminal and the log file when all families are listed.

The \_famfrom saves the information about font type foundry or manufacturer or designer or license owner. You can use it before \_faminfo to print \_famfrom info into the catalog. The \_famfrom data is applied to each following declared families until new \_famfrom is given. Use \_famfrom {} if the information is not known.

\subsection{Notices about extension of \font primitive}

Unicode fonts are loaded by extended \font primitive. This extension is not activated in OpTEX by default, \texttt{\initunifonts} macro activates it. You need not use \texttt{\initunifonts} explicitly if \texttt{\fontfam} macro is used because \texttt{\fontfam} runs it internally.

The \texttt{\initunifonts} loads the Lua code from the Luaotfload package which implements the \font primitive extension. See its documentation \texttt{luaotfload-latex.pdf} for information about all possibilities of extended \font primitive.

The OpTEX format is initialized by \texttt{lualatex} engine by default but you can initialize it by \texttt{lualhtex} engine too. Then the harfbuzz library is ready to use for font rendering as an alternative to built-in
font renderer from Luaotfload. The harfbuzz library gives more features for rendering Indic and Arabic scripts. But it is not used as default, you need to specify `mode=harf` in the fontfeatures field when `\font` is used. Moreover, when `mode=harf` is used, then you must specify `script` too. For example

```
\font\devafont=[NotoSansDevanagari-Regular]:mode=harf;script=dev2
```

If the `luahbtex` engine is not used then `mode=harf` is ignored. See Luaotfload documentation for more information.

### 2.13.15 Implementation of the Font Selection System

The variant selectors `\rm`, `\bf`, `\it`, `\bi`, `\tt` are defined (roughly speaking) by

```
\def\<\times\>\{\_\texttt{tryload}(\times)\}_{\texttt{ten}(\times)}
```

where `(\times)` is “internal variant name” `\rm` or `\bf` or `\it` or `\bi` or `\tt`. There are five “internal font switchers” `\tenrm`, `\tenbf`, `\tenit`, `\tenbi` and `\tentt`. They are used almost for all fonts selected by the Fonts Selection System. For example, `\tenbf` is the switcher for bold variant of the current family in the current font context. The `\bf` macro is defined as `\texttt{tryloadbf} \_\texttt{tenbf}`. If the font context (font family, font size, features) is not changed, then `\texttt{tryloadbf}` is `\relax` and `\tenbf` font switcher selects given font. If the font context is changed, then `\texttt{tryloadbf}` is re-defined (see `\texttt{reloading}` macro) to load new bold variant of the font using `\texttt{resizesfont}`. The loaded font is saved to `\tenbf` switcher and `\texttt{tryloadbf}` returns back to the `\relax` meaning. So, `\bf` macro loads new font with current font context and then selects it by `\tenbf` selector. The word “ten” is used here only for historical reason; the font can be at arbitrary size.

The `\texttt{reloading}` macro is run whenever font context is changed. It activates `\texttt{tryload}(\times)` for `(\times)` in `\rm`, `\bf`, `\it` and `\bi`. The `\texttt{load}(\times)\_{\texttt{ten}(\times)}` is processed for this.

The `\texttt{tryloadtt}` is implemented differently because we want to keep family independence for `\tt` macro, see section 2.13.7. So, `\texttt{tryloadtt}` is defined constantly as “loading `\tt` font” and it is not re-defined to `\relax`. On the other hand, `\texttt{tryloadtt}` is re-defined in the `\texttt{initunifonts}` macro or when `\texttt{famvardef}``tt` is used.

### Fonts Selection System

The Font Selection system allows to use `\currvar` instead of an explicitly specified variant selector. The current variant is extracted from `\the\font` output which could be the `\ten(\times)` control sequence. Then `\currvar` expands to `\_\texttt{rm}` or `\_\texttt{it}` etc.

The `\texttt{resizesfont}` `{\langle variant-name \rangle}`{\langle font switch \rangle} is the heart of the Fonts Selection System. It resizes the font given by the variant with respect to the current font context and sets a new `{\langle font-switch \rangle}`. The `{\langle variant-name \rangle}` is `\rm` or `\bf` or `\it` or `\bi` or `\tt`. The new `{\langle font-switch \rangle}` is declared (roughly speaking) by:

```
\font {font switch} = {\langle fontname of \rangle}\_\texttt{ten}(\langle variant-name \rangle) \_\texttt{sizeselect} % in TFM mode
\font {font switch} = {\langle fontname of \rangle}\_\texttt{sizeselect} % in UTF mode
```

The font is loaded by `\texttt{doresizesfont}`{\langle font switch \rangle}. This macro has meaning `\texttt{doresizesunifont}` in TFM mode (default in format) and it switches to `\texttt{doresizesunifont}` when `\texttt{initunifonts}` is used. The `\texttt{fontname of}` is generated by the `\texttt{fontname TpX primitive where \_\texttt{rfontskipat} removes the
at\{dimen\} part of the \fontname output.

The \_whatresize is defined as \{variant-name\}.

The \fontloaded\{font switch\} is a macro which can be used for post-processing when a font is loaded.

\_doresizeunifont\{font-switch\} implements the OTF mode of loading fonts \_doresizefont. There is a fallback to TFM mode if \_fontnamegen is not defined.

The \_fontnamegen expands to the font name/file:font-features depending on the current font context.

If a font is loaded by \_resizefont or \resizethefont then the \fontloaded\{font switch\} is called immediately after it. If the font is loaded first then its \skewchar is equal to \(-1\). We run \_newfontloaded\{font switch\} and set \skewchar=-2 in this case. A user can define a \_newfontloaded macro. We are sure that \_newfontloaded macro is called only once for each instance of the font given by its name, OTF features and size specification. The \skewchar value is globally saved to the font (like \fontdimen). If it is used in math typesetting then it is set to a positive value.

The \_newfontloaded should be defined for micro-typographic configuration of fonts, for example. See OpTEX trick 0058.

\initunifonts macro extends Lua\TeX’s font capabilities, in order to be able to load Unicode fonts. Unfortunately, this part of Op\TeX depends on the luaotfload package, which adapts Con\TeXt’s generic font loader for plain \TeX and \LaTeX. luaotfload uses Lua functions from \LaTeX’s luatexbase namespace, we provide our own replacements. Moreover, \initunifont switches with the \_doresizefont macro to OTF mode which is represented by the macro \_doresizeunifont. Finally, \initunifonts sets itself to relax because we don’t want to do this work twice.

\_ttunifont is default font for \tt variant. User can re-define it or use \famvardef\tt.
The \famdecl \{\family\} \{\selector\} \{\comment\} \{\modifiers\} \{\variants\} \{\math\} \{\font for testing\} \{\def\fontnamegen\{\data\}\} runs \initunifonts, then checks if \selector is defined. If it is true, then closes the file by \endinput. Else it defines \selector and saves it to the internal \f:currfamily:main.fam command. The macro \initfontfamily needs it. The \currfamily is set to the \selector because the following \moddef commands need to be in the right font family context. The \currfamily is set to the \selector by the \selector too, because \selector must set the right font family context. The font family context is given by the current \currfamily value and by the current meaning of the \fontnamegen macro. The \mathfaminfo is saved for usage in the catalog.

\fvars (rm-template) \{bf-template\} \{it-template\} \{bi-template\} saves data for usage by the \currV macro. If a template is only dot then previous template is used (it can be used if the font family doesn’t dispose with all standard variants).

\currV expands to a template declared by \fvars depending on the \variant name. Usable only of standard four variants. Next variants can be declared by the \famvardef macro.
\csetV \{key\}={\value},...,\{key\}={\value} expands to \def\key{\value} in the loop.
\onlyif \{key\}={\value-a},\{value-b\},...,\{value-z\}: \{\what\} runs \what only if the \key is defined as \value-a or \value-b or ... or \value-z.
\precommalist \{a\},cd,\end, expands to \{a\},cd, (auxiliary macro used in \onlyif).
\ffonum is a shortcut for oldstyle digits font features used in font family files. You can do \let\ffonum=\ignoreit if you don’t want to set old digits together with \caps.

The \moddef \{\modifier\} \{\data\} simply speaking does \def\modifier\{\data\}, but we need to respect the family context. In fact, \protected\def\f:\{current family\}:\{\modifier\} \{\data\} is performed and the \modifier is defined as \famdepend \{\modifier\} \f:\currfamily:\{\modifier\}. It expands to
\famvardefA \moddef \famdependent \fontdef \fontswitch \fontprimitive \font \fontswitch \fontprimitive \font
\fontswitch\fontprimitive \fontprimitive \fontprimitive \font \font \font \font \fontswitch \fontprimitive \fontprimitive \fontprimitive \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \font \fo
\def \_tryload:{currfam}:⟨XX⟩ {\fontdef \_ten:⟨XX⟩ {⟨data⟩}} loads font \_ten⟨XX⟩,
\protected\def \langle XX⟩ {\famdepend \langle XX⟩ {f:\langle currfam⟩:⟨XX⟩}},
\def \_f:(currfam):(⟨XX⟩) {\_tryload:{currfam}:⟨XX⟩ \_ten:⟨XX⟩} keeps family dependent definition,
\def \_currvar:-ten:⟨XX⟩ {\langle XX⟩} in order to the \currvar macro work correctly.
\famvardef \tt behaves somewhat differently: it doesn’t re-define the \tt macro which is defined as \_tryloadtt \_tentt in sections 2.14 and 2.16.2. It only re-defines the internal \_tryloadtt macro. Note, that you cannot use \tt inside \famvardef\tt. So, new \tt macro does not load \_ttunifont but uses font from a standard variant rm, bf, it or bi with given font context.

The \fontfam [{(Font Family)}] does:
- Convert its parameter to lower case and without spaces, e.g. ⟨fontfamily⟩.
- If the file f-{⟨fontfamily⟩}.opm exists read it and finish.
- Try to load user defined \fams-local.opm.
- If the ⟨fontfamily⟩ is declared in \fams-local.opm or \fams-ini.opm read relevant file and finish.
- Print the list of declared families.

The \fams-local.opm is read by the \_tryloadfamslocal macro. It sets itself to \_relax because we need not load this file twice. The \_listfamnames macro prints registered font families to the terminal and to the log file.
When the `fams-ini.opm` or `fams-local.opm` files are read then we need to save only a mapping from family names or alias names to the font family file names. All other information is ignored in this case. But if these files are read by the \_listfamnames macro or when printing a catalog then more information is used and printed.

\_famtext does nothing or prints the text on the terminal.

\_faminfo \{Family Name\} \{\{comments\}\} \{\{file-name\}\} \{\{mod-plus-vars\}\} does \def \_famf:\{familyname\} \{\{file-name\}\} or prints information on the terminal.

\_famalias \{Family Alias\} does \def \_famf:\{familyalias\} \{\{file-name\}\} where \{\{file-name\}\} is stored from the previous \_faminfo command. Or prints information on the terminal.

\_famfrom declares type foundry or owner or designer of the font family. It can be used in `fams-ini.opm` or `fams-local.opm` and it is printed in the font catalog.

When the \fontfam\[catalog\] is used then the file `fonts-catalog.opm` is read. The macro \_faminfo is redefined here in order to print catalog samples of all declared modifiers/variant pairs. The user can declare different samples and different behavior of the catalog, see the end of catalog listing for more information. The default parameters \catalogsample, \catalogmathsample, \catalogonly and \catalogexclude of the catalog are declared here.

The font features are managed in the \fontfeatures macro. It expands to

- \_defaultfontfeatures – used for each font,
- \_ffadded – features added by \setff,
- \_ffcolor – features added by \setfontcolor,
- \_fletterspace – features added by \setletterspace,
- \_fwordspace – features added by \setwordspace.

The macros \_ffadded, \_ffcolor, \_fletterspace, \_fwordspace are empty by default.

The \setff \{\{features\}\} adds next font features to \_ffadded. Usage \setff{} resets empty set of all \_ffadded features.
The \texttt{\setfontcolor} and \texttt{\setletterspace} are macros based on the special font features provided by \LaTeX{} (and by \TeX{} too but it is not our business). The \texttt{\setwordspace} recalculates the \texttt{\fontdimen2,3,4} of the font using the \texttt{\_setwsp} macro which is used by the \texttt{\_doresizeunifont} macro. It activates a dummy font feature \texttt{\_Ws} too in order the font is reloaded by the \texttt{\font} primitive (with independent \texttt{\fontdimen} registers).

\texttt{\_regoptsizes} \texttt{\langle internal-template \rangle \langle left-output?\rangle \langle right-output?\rangle \langle resizing-data \rangle \langle mid-output?\rangle <\langle size \rangle>} in the loop.

\texttt{\_optfontalias \{⟨template A⟩\} \{⟨template B⟩\}} is defined as \texttt{\let \_oz:⟨template A⟩=\_oz:⟨template B⟩}.

2.14 Preloaded fonts for math mode

The Computer Modern and AMS fonts are preloaded here in classical math-fam concept, where each math family includes three fonts with max 256 characters (typically 128 characters).
On the other hand, when \texttt{\fontfam} macro is used in the document then text font family and appropriate math family is loaded with Unicode fonts, i.e. Unicode-math is used. It re-defines all settings given here.

The general rule of usage the math fonts in different sizes in Op\TeX\ says: set three sizes by the macro \texttt{\setmathsizes \[\langle text-size\rangle/\langle script-size\rangle/\langle scriptscript-size\rangle\]} and then load all math fonts in given sizes by \texttt{\normmath} or \texttt{\boldmath} macros. For example

\texttt{\setmathsizes[12/8.4/6]\normmath ... math typesetting at 12 pt is ready.}

We have two math macros \texttt{\normmath} for the normal shape of all math symbols and \texttt{\boldmath} for the bold shape of all math symbols. The second one can be used in bold titles, for example. These macros load all fonts from all given math font families.

The classical math family selectors \texttt{\mit}, \texttt{\cal}, \texttt{\bbchar}, \texttt{\frak} and \texttt{\script} are defined here. The \texttt{\rm}, \texttt{\bf}, \texttt{\it}, \texttt{\bi} and \texttt{\tt} does two things: they are variant selectors for text fonts and math family selectors for math fonts. The idea was adapted from plain \TeX. These macros are redefined when \texttt{unimat-codes.opm} is loaded, see the section 2.16.2.
The optical sizes of Computer Modern fonts, AMS, and other fonts are declared here.

\protected\def \bbchar \{ \fam5 \} % double stroked letters
\protected\def \frak \{ \fam7 \} % fraktur
\protected\def \script \{ \fam6 \} % more extensive script than \cal

\public \rm \bf \it \bi \tt \mit \cal \bbchar \frak \script ;

\% CM math fonts, optical sizes:
\_regtfm cmmi 0 cmmi5 5.5 cmmi6 6.5 cmmi7 7.5 cmmi8 8.5 cmmi9 9.5
\% cmr
\_regtfm cmr 0 cmr5 5.5 cmr6 6.5 cmr7 7.5 cmr8 8.5 cmr9 9.5 cmr10 11.1 cmr12 15
\_regtfm cmsy 0 cmsy5 5.5 cmsy6 6.5 cmsy7 7.5 cmsy8 8.5 cmsy9 9.5 cmsy10 *
\_regtfm cmex 0 cmex7 7.5 cmex8 8.5 cmex9 9.5 cmex10 *
\_loadmathfamily \langle number \rangle \{ font \} loads one math family, i.e. the triple of fonts in the text size, script size and script-script size. The \{ font \} is \{ font-id \} used in the \_regtfm parameter or the real TFM name. The family is saved as \{ fam \langle number \rangle \}.

\setmathfamily \langle number \rangle \{ font-switch \} \_setmathfamily \langle number \rangle \{ font-switch \} loads one math family like \loadmathfamily does it. But the second parameter is a \{ font-switch \} declared previously by the \font primitive. The font family is loaded at \_sizemtext, \_sizemscript and \_sizemsscript sizes. These sizes are set by the \_setmathsizes \[ \langle text-size \rangle / \langle script-size \rangle / \langle scriptscript-size \rangle \] macro. These parameters are given in the \ptmunit unit, it is set to 1pt and it is set to 1pt by default.

\_corrmsize \langle factor \rangle \langle space \rangle can be used just before \loadmathfamily or \setmathfamily. The \langle factor \rangle is decimal number, it denotes scale-factor “size of loaded math font in \textstyle : size of text font”. You can use it in \normalmath or \boldmath macros if you want to do a corrections (for example due to different ex-height in text and math font). The \corrmsize is applied only to one following \loadmathfamily or \setmathfamily. If it is missing then the \langle factor \rangle is 1 for such math family (i.e. no size corrections).
The \texttt{\_setmathdimens} macro is used in \texttt{\normalmath} or \texttt{\boldmath} macros. It makes math dimensions dependent on the font size (plain TeX sets them only for 10pt typesetting). The \texttt{\skewchar} of some math families are set here too.

Finally, we preload a math fonts collection in [10/7/5] sizes when the format is generated. This is done when \texttt{\_suppressfontnotfounderror=1} because we need not errors when the format is generated. Maybe there are not all fonts in the TeX distribution installed.

\subsection*{2.15 Math macros}

The category code of the character \texttt{\_} remains as the letter (11) and the mathcode of it is \texttt{"8000}. It means that it is an active character in math mode. It is defined as the subscript prefix.

There is a problem: The \texttt{x\_n} is tokenized as \texttt{x}, \texttt{\_}, \texttt{n} and it works without problems. But \texttt{\int\_a}^b is tokenized as \texttt{\int\_a}^\texttt{b}. The control sequence \texttt{\int\_a} isn’t defined. We must write \texttt{\int\ _a^b}.

The Lua code presented here solves this problem. But you cannot set your own control sequence in the form \texttt{\langle word\_\langle nonletter\rangle} because such control sequences are inaccessible: preprocessor rewrites it.

The \texttt{\mathsbon} macro activates the rewriting rule \texttt{\langle word\_\langle nonletter\rangle} to \texttt{\langle word\_\langle nonletter\rangle} and \texttt{\langle word\_\langle letter\rangle\langle nonletter\rangle} to \texttt{\langle word\_\langle letter\rangle\langle nonletter\rangle} at input processor level. The \texttt{\mathsboff} deactivates it. You can ask by \texttt{\_ifmathsb} if this feature is activated or deactivated. By default, it is activated in the \texttt{\everyjob}, see section 2.1. Note, that the \texttt{\everyjob} is processed after the first line of the document is read, so the \texttt{\mathsbon} is activated from the second line of the document.
All mathcodes are set to equal values as in plain TeX. But all encoding-dependent declarations (like these) will be set to different values when a Unicode-math font is used. 

\mathcodes
\mathcode``\@``=2201 % \cdot
\mathcode``\^^A``=3223 % \downarrow
\mathcode``\^^B``=010B % \alpha
\mathcode``\^^C``=010C % \beta
\mathcode``\^^D``=225E % \land
\mathcode``\^^E``=023A % \lnot
\mathcode``\^^F``=3232 % \in
\mathcode``\^^G``=0119 % \pi
\mathcode``\^^H``=0115 % \lambda
\mathcode``\^^I``=010D % \gamma
\mathcode``\^^J``=010E % \delta
\mathcode``\^^K``=3222 % \uparrow
\mathcode``\^^L``=2206 % \pm
\mathcode``\^^M``=2208 % \oplus
\mathcode``\^^N``=0231 % \infty
\mathcode``\^^O``=0140 % \partial
\mathcode``\^^P``=321A % \subset
\mathcode``\^^Q``=321B % \supset
\mathcode``\^^R``=225C % \cap
\mathcode``\^^S``=225B % \cup
\mathcode``\^^T``=0238 % \forall
\mathcode``\^^U``=0239 % \exists
\mathcode``\^^V``=220A % \otimes
\mathcode``\^^W``=3224 % \leftrightarrow
\mathcode``\^^X``=3221 % \leftarrow
\mathcode``\^^Y``=3220 % \rightarrow
\mathcode``\^^Z``=8000 % \neq
\mathcode``\^^[``=2205 % \diamond
\mathcode``\^^\``=303D % \equiv
\mathcode``\^^^``=3211 % \leq
\mathcode``\^^_``=225F % \lor
\mathcode``\^^\``=8000 % \space
\mathcode``\@``!*5021
\mathcode``\@``!*5020
\mathcode``\@``!*5029
\mathcode``\@``!*2203 % \ast
\mathcode``\@``!*202B
\mathcode``\@``!*613B
\mathcode``\@``!*2200
\mathcode``\@``!*013A
\mathcode``\@``!*013D
\mathcode``\@``!*303A
\mathcode``\@``!*603B
\mathcode``\@``!*313C
\mathcode``\@``!*303D
\mathcode``\@``!*313E
\mathcode``\@``!*503F
All control sequences declared by \mathchardef are supposed (by default) only for public usage. It means that they are declared without _ prefix. If such sequences are used in internal OpTiX macro then their internal prefixed form is declared using \private macro.

These encoding dependent declarations will be set to different values when Unicode-math font is loaded. The declared sequences for math symbols are not hyperlinked in this documentation.

The math functions like log, sin, cos are declared in the same way as in plainTEX, but they are \protected in OpTiX.
These macros are defined similarly as in plain\TeX. Only internal macro names from plain\TeX with \@ character are re-written in a more readable form. \sp is an alternative for ^ and \sb alternative for _ was defined at line 27 of the file math-macros.opm.

Active \prime character is defined here.

Math relations defined by the \jointrel plain \TeX macro:
\protected\def\hookleftarrow{\_leftarrow\_joinrel\_rhook}
\protected\def\bowtie{\_mathrel\triangleright\_joinrel\mathrel\triangleleft}
\protected\def\models{\_mathrel\|\_joinrel\=}
\protected\def\Longrightarrow{\_Relbar\_joinrel\Rightarrow}
\protected\def\longrightarrow{\_relbar\_joinrel\rightarrow}
\protected\def\longleftarrow{\_leftarrow\_joinrel\relbar}
\protected\def\Longleftarrow{\_Leftarrow\_joinrel\Relbar}
\protected\def\longmapsto{\_mapstochar\longrightarrow}
\protected\def\longleftrightarrow{\_leftarrow\_joinrel\rightarrow}
\protected\def\Longleftrightarrow{\_Leftarrow\_joinrel\Rightarrow}
\protected\def\iff{\_thicksk\_Longleftrightarrow\_thicksk}
\protected\def\ldots{\_mathinner{\_ldotp\_ldotp\_ldotp}}
\protected\def\cdots{\_mathinner{\_cdotp\_cdotp\_cdotp}}
\protected\def\vdots{\_vbox{\_baselineskip=.4em \_lineskiplimit=\_zo \kern.6em \hbox{.}\hbox{.}\hbox{.}}}
\protected\def\ddots{\_mathinner{\_mkern1mu\raise.7em\vbox{\_kern.7em\hbox{.}}\_mkern2mu \raise.4em\hbox{.}\_mkern2mu\raise.1em\hbox{.}\_mkern1mu}}
\protected\def\adots{\_mathinner{\_mkern1mu\raise.1em\hbox{.}\_mkern2mu \raise.4em\hbox{.}\_mkern2mu\raise.7em\vbox{\_kern.7em\hbox{.}}\_mkern1mu}}
\protected\def\acute{\_mathaccent"7013} \grave{\_mathaccent"7012} \ddot{\_mathaccent"707F} \tilde{\_mathaccent"707E} \bar{\_mathaccent"7016} \breve{\_mathaccent"7015} \check{\_mathaccent"7014} \hat{\_mathaccent"705E} \vec{\_mathaccent"017E} \dot{\_mathaccent"705F} \widetilde{\_mathaccent"0365} \widehat{\_mathaccent"0362}
\def\math{
\mathsurround\zo}
\protected\def\skew #1#2#3{{\_muskip0=#1mu\divide\muskip0=by2 \_mkern\muskip0 #2{\_mkern\muskip0(#3)}\_mkern\muskip0}}
\protected\def\overrightarrow #1{\vbox{\_math\ialign{##\_crcr
\rightarrowfill\crcr\noalign{\kern-.1em \nointerlineskip}$\_hfil\displaystyle{#1}\_hfil$\crcr}}}
\protected\def\overleftarrow #1{\vbox{\_math\ialign{##\_crcr
\leftarrowfill\crcr\noalign{\kern-.1em \nointerlineskip}$\_hfil\displaystyle{#1}\_hfil$\crcr}}}
\protected\def\overbrace #1{\mathop{\_\mskip-1mu\vrule width 3\zop\vbox{\_\kern-1mu\strut\hfil\_\displaystyle{#1}\hfil\kern-1mu\strut\_\mskip-1mu\vrule width 3\zop}}}
\protected\def\underbrace #1{\mathop{\_\mskip-1mu\vrule width 3\zop\vbox{\_\kern-1mu\strut\hfil\_\displaystyle{#1}\hfil\kern-1mu\strut\_\mskip-1mu\vrule width 3\zop}}}
\protected\def\maths#1#2#3{#1\_\math#2#3}
\protected\def\ske#1#2#3{#1\_\skew#2#3{#3}}
\protected\def\overrightarrow #1{\_\overrightarrow{#1}}
\protected\def\overleftarrow #1{\_\overleftarrow{#1}}
\protected\def\overbrace #1{\_\overbrace{#1}}
\protected\def\underbrace #1{\_\underbrace{#1}}
\protected\def\overrightarrow #1{\_\overrightarrow{#1}}
\protected\def\overleftarrow #1{\_\overleftarrow{#1}}
\protected\def\overbrace #1{\_\overbrace{#1}}
\protected\def\underbrace #1{\_\underbrace{#1}}
\def\makesmash#1{\setbox0=\hbox{#1}\finsmash}
\def\mathsmash#1#2{\setbox0=\hbox{$\math#1{#2}$}\finsmash}
\def\finsmash{\ht0=\zo\dp0=\zo\box0}
\public \mathpalette \vphantom \hphantom \phantom \mathstrut \smash ;
\cong, \notin, \rightleftharpoons, \dot{\text{e}}, \doteq, \bmod and \pmod macros from plain \TeX.

\protected\def cong{\mathrel{\mathpalette\overeq\sim}} % congruence sign
\def overeq#1#2{\lower.05em\vbox{\lineskiplimit\maxdimen\lineskip=-.05em
\ialign{$\math#1\hfil##\hfil$\crcr#2\crcr=\crcr}}}
\protected\def\notin{\mathrel{\mathpalette\cancel\in}}
\def\cancel#1#2{\math\ooalign{$\hfil#1\mkern1mu/\hfil$\crcr$#1#2$}}
\protected\def\rightleftharpoons{\mathrel{\mathpalette\rlhp{}}}
\def\rlhp#1{\vcenter{\math\hbox{\ooalign{\raise.2em\hbox{$#1\rightharpoonup$}\crcr
$#1\leftharpoondown$}}}}
\protected\def\buildrel#1\over#2{\mathrel{\mathop{\kern\zo #2}\limits^{#1}}}
\protected\def\doteq{\buildrel\textstyle.\over=}
\in \sim ;
\public \cong \notin \rightleftharpoons \doteq \bmod \pmod ;
\matrix and \pmatrix behave as in Plain \TeX, if it is used in the \displaystyle. On the other
hand, it is printed in smaller size (by appropriate amount) in \textstyle = \scriptstyle and
\scriptscriptstyle. This feature is new in Op\TeX.

\protected\def\matrix#1{\null\thinsk\edef\tmpa{\the\numexpr\mathstyle/4\relax}% 0 0 1 1 1 1 2 2
\vcenter{\matrixbaselines\math\ialign{\the\lmfil$\matrixstyle##\hfil&&\quad\the\lmfil$\matrixstyle##\hfil\crcr
\mathstrut\crcr\noalign{\kern-\baselineskip}\phantom{#1}\crcr\mathstrut\crcr\noalign{\kern-\baselineskip}}}
\thinsk}
\protected\def\pmatrix#1{\left(\matrix{#1}\right)}
\cases and \bordermatrix macros are almost identical as in plain \TeX. You can simply re-define
\bordermatrix with other delimiters using the common \bordermatrixwithdelims macro.
The \texttt{\textbackslash halign} macro behaves like in Plain \TeX{} by default. It creates the \texttt{\textbackslash vcenter} in the math mode.

The content is two column \texttt{\textbackslash halign} with right-aligned left column and left-aligned right column. The table items are in \texttt{\textbackslash displaystyle} and the \texttt{\textbackslash baselineskip} is advanced by \texttt{\textbackslash jot} (3pt in plain \TeX{}). It follows from the default settings of \texttt{\textbackslash eqlines} and \texttt{\textbackslash eqstyle} parameters.

In Op\TeX{}, this macro is more flexible. See section 4.4 in the \textit{Typesetting Math with Op\TeX{}}. The \texttt{\textbackslash baselineskip} value is set by the \texttt{\textbackslash eqlines} parameter and math style by the \texttt{\textbackslash eqstyle} parameter.

There are more possible columns than two (used in classical Plain \TeX{}); \texttt{\textbackslash rlcrlcrlc} etc. where \texttt{\textbackslash r} and \texttt{\textbackslash l} columns are without spaces and \texttt{\textbackslash c} column (if used) has space \texttt{\textbackslash eqspace}/2 at its both sides.

\texttt{\textbackslash displaylines} \{\texttt{\textbackslash formula\textbackslash cr\textbackslash formula\textbackslash cr...\textbackslash formula}\} creates horizontally centered formulae. It behaves exactly as in Plain \TeX{}. The \texttt{\textbackslash halign} is applied directly in the outer display environment with lines of type \texttt{\textbackslash hbox to\displaywidth}. This enables to break lines inside such display to more pages but it is impossible to use \texttt{\textbackslash eqno} or \texttt{\textbackslash leqno} or \texttt{\textbackslash eqmark}.

Op\TeX{} offers \texttt{\textbackslash displaylines to\{\texttt{\textbackslash dimen}\}\texttt{\textbackslash formula\textbackslash cr\textbackslash formula\textbackslash cr...\textbackslash formula}\}} as an alternative case of usage \texttt{\textbackslash displaylines}. See section 4.3 in the \textit{Typesetting Math with Op\TeX{}}. The centered formulas are in \texttt{\textbackslash vcenter} in this case, so lines cannot be broken into more pages, but this case enables to use \texttt{\textbackslash eqno} or \texttt{\textbackslash leqno} or \texttt{\textbackslash eqmark}.

\texttt{\textbackslash openup, \textbackslash eqalignno} and \texttt{\textbackslash leqalignno} macros are copied from Plain \TeX{} unchanged.
These macros are inspired by ams-math.tex file.

\begin{verbatim}
\def\amsafam{4} \def\amsbfam{5}
\mathchardef \boxdot "2\amsafam 00
\mathchardef \boxplus "2\amsafam 01
\mathchardef \boxtimes "2\amsafam 02
\mathchardef \square "0\amsafam 03
\mathchardef \blacksquare "0\amsafam 04
\mathchardef \centerdot "2\amsafam 05
\mathchardef \lozenge "0\amsafam 06
\mathchardef \blacklozenge "0\amsafam 07
\mathchardef \circlearrowright "3\amsafam 08
\mathchardef \circlearrowleft "3\amsafam 09
\mathchardef \rightleftharpoons "3\amsafam 0A
\mathchardef \leftrightharpoons "3\amsafam 0B
\mathchardef \boxminus "2\amsafam 0C

\end{verbatim}

...etc. (see math-macros.opm)

The \not macro is re-defined to be smarter than in plain 
\TeX. The macro follows this rule:

\begin{verbatim}
\not< becomes \_nless
\not> becomes \_ngtr
if \_notXXX is defined, \not\XXX becomes \_notXXX;
if \_nXXX is defined, \not\XXX becomes \_nXXX;
otherwise, \not\XXX is done in the usual way.
\end{verbatim}

\begin{verbatim}
\protected\def \_not#1{\ifx #1<\_nless \else\ifx #1>\_ngtr \else\edef\tmpn{\csstring#1}\ifcsname _not\tmpn\endcsname \csname _not\tmpn\endcsname \else \ifcsname _n\tmpn\endcsname \csname _n\tmpn\endcsname \else \mathrel{\mathord{\notchar}\mathord{#1}}\fi \fi \fi \fi}
\end{verbatim}

\begin{verbatim}
\mathstyles{{\langle math list\rangle}} behaves like {\langle math list\rangle}, but you can use the following commands in the {\langle math list\rangle}:
\begin{itemize}
  \item \currstyle which expands to \displaystyle, \textstyle, \scriptstyle or \scriptscriptstyle depending on the current math style when \mathstyles was opened.
  \item \dobystyle{\langle D\rangle}{\langle T\rangle}{\langle S\rangle}{\langle SS\rangle} is expandable macro. It expands to \langle D\rangle, \langle T\rangle, \langle S\rangle or \langle SS\rangle depending on the current math style when \mathstyles was opened.
  \item The value of the \stylenum is 0, 1, 2 or 3 depending on the current math style when \mathstyles was opened.
\end{itemize}

Example of usage of \mathstyles: \def\mathframe#1{\mathstyles{\frame{\currstyle{#1}}}}.

\begin{verbatim}
\newcount\stylenum
\def\mathstyles#1{{\mathchoice{\stylenum0 #1}{\stylenum1 #1}{}{}}}
\end{verbatim}

\begin{verbatim}
The \cramped macro sets the cramped variant of the current style. Note that \currstyle initializes non-cramped variants. The example \mathframe above should be:
\end{verbatim}

90
Second note: \cramped macro reads the current math style from the \mathstyle Lua\TeX\ primitive, so it does not work in numerators of generalized fractions but you can use it before the fraction is opened: $\cramped \left(\frac{x^2}{y^2}\right)$.

\input math-macros.opm

The \mathbox{⟨text⟩} macro is copied from OPmac trick 078. It behaves like \hbox{⟨text⟩} but the ⟨text⟩ is scaled to a smaller size if it is used in scriptstyle or scriptscript style. The \_textmff and \_scriptmff are redefined in order to respect optical sizes. If we are in script style then the math mode starts in text style, but optical size is given to script style. The \mathbox in non-Unicode math respects optical sizes using different principle.

\input math-macros.opm

2.16 Unicode-math fonts

The \loadmath \{⟨Unicode-math font⟩\} macro loads math fonts and redefines all default math-codes using \input unimath-codes.opm. If Unicode-math font is loaded then \_mathloadingfalse is set, so the new Unicode-math font isn’t loaded until \doloadmath is used.

\loadboldmath \{⟨bold-font⟩\} \to \{⟨normal-font⟩\} loads bold variant only if ⟨normal-font⟩ was sucessfully loaded by the previous \loadmath. For example:

\loadmath \{[xitsmath-regular]\}
\loadboldmath \{[xitsmath-bold]\} \to \{[xitsmath-regular]\}

There are very few Unicode-math fonts with full \boldmath support. I know only XITSMath-Bold and KpMath-Bold. If \loadboldmath is not used then “faked bold” created from \normalmath is used by default.

The \loadmath macro was sucessfully tested on:

\loadmath[XITSMath-Regular] \ldots XITS MATH
\loadmath[Latinmodern-math] \ldots Latin Modern Math
\loadmath[TeXGyreTermes-math] \ldots TeXGyre Termes Math
\loadmath[TeXGyreBonum-math] \ldots TeXGyre Bonum Math
\loadmath[TeXGyrePagella-math] \ldots TeXGyre Pagella Math
\loadmath[TeXGyreSchola-math] \ldots TeXGyre Schola Math
\loadmath[TeXGyreDejaVu-math] \ldots TeXGyre DeJaVu Math
\loadmath[LibertinusMath-Regular] \ldots Libertinus Math
\loadmath[FiraMath-Regular] \ldots Fira Math
\loadmath[Asana-Math] \ldots Asana Math
\loadmath[KpMath-Regular] \ldots KP fonts Math

2.16.1 Unicode-math macros preloaded in the format

\input math-unicode.opm

\loadmath \{⟨Unicode-math font⟩\} loads the given font. It does:

\begin{itemize}
\item define \_unimathfont as ⟨Unicode-math font⟩,
\item redefine \normalmath and \boldmath macros to their Unicode counterparts,
\item load the \_unimathfont by \normalmath,
\item print information about the loaded font on the terminal,
\item redefine all encoding dependent setting by \input unimath-codes.opm,
\item protect new loading by setting \_ifmathloading to false.
\end{itemize}
\noloadmath disallows Unicode-math loading by \_mathloadingfalse. \\
doloadmath allows Unicode-math loading by \_mathloadingtrue.
If you try the example above about `\loadboldmath{[xitsmath-bold]} \to {[xitsmath-regular]}`, then you can find a bug in XITSMath-Bold font: the symbols for norm ‖x‖ are missing. So, we have to define `\boldmath` macro manually. The missing symbol is loaded from family 5 as no-bold variant in our example:

```
\loadmath{[xitsmath-regular]}
\def\boldmath{%
  \loadumathfamily 1 {[xitsmath-bold]} \% Base font
  \loadumathfamily 4 refs \% script
  \loadumathfamily 5 {[xitsmath-regular]} \%
  \def\{|{\textstyle \Udelimiter 0 5 "02016 }% \% norm delimiter from family 5
  \setmathdimens}
}
```

`\loadumathfamily ⟨number⟩ ⟨{font}⟩⟨{font features}⟩` loads the given Unicode-math fonts in three sizes using single ⟨{font}⟩ with different \texttt{mathsize=1,2,3} font features. The math font family is set with given ⟨{number}⟩. The ⟨{font features}⟩ are added to the default `\mfontfeatures` and to the size-dependent features {\texttt{ssty=1} if script size is asked or \texttt{ssty=2} if scriptscriptsize is asked.

`\mparams⟨number⟩` inserts additional font feature \texttt{nomathparam} if the ⟨{number}⟩ of the family is greater than 3. Lua\TeX\ inserts math parameters (thickness of fraction rules etc., see section 7.4 in Lua\TeX\ documentation) repeatedly from loaded math fonts if \texttt{nomathparam} is not given. We want to load these parameters only from fonts at families 0-3 (and actually we are using only family 1 as main math font). The `\corrmsize ⟨factor⟩⟨space⟩` can be used just before `\loadumathfamily`, see section 2.14 for more information.

The `\textmff`, `\scriptmff` and `\sscriptmff` are font features for text, script and script sizes respectively. They are locally re-defined in `\mathtt{mathbox}` macro.

Unicode math font includes all typical math alphabets together, user needs not to load more \TeX\ math families. These math alphabets are encoded by different parts of Unicode table. We need auxiliary macros for setting mathcodes by selected math alphabet.

\texttt{\umathrange} \{⟨from⟩−⟨to⟩\}⟨{class}⟩⟨{family}⟩⟨{first}⟩ sets \texttt{Mathcodes} of the characters in the interval \{⟨from⟩−⟨to⟩\} to \{⟨first⟩\}, \{⟨first⟩+1\}, \{⟨first⟩+2\} etc., but \texttt{\umathcharholes} are skipped (\texttt{\umathcharholes} are parts of the Unicode table not designed for math alphabets but they cause that the math alphabets are not continuously spread out in the table; I mean that the designers were under the influence of drugs when they created this part of the Unicode table). The \{⟨from⟩−⟨to⟩\} clause includes normal letters like A-Z.

\texttt{\umathrangegreek} \{⟨first⟩\} is the same as \texttt{\umathrange} \{⟨alpha⟩−⟨omega⟩\}⟨{first}⟩.

\texttt{\greekdef} \{control sequences} \{\relax\} defines each control sequence as a normal character with codes \texttt{\umathnumB}, \texttt{\umathnumB+1}, \texttt{\umathnumB+2} etc. It is used for redefining the contol sequences for math Greek \texttt{\alpha}, \texttt{\beta}, \texttt{\gamma} etc.
2.16.2 Macros and codes set when \loadmatfont is processed

The file \texttt{unimath-codes.opm} is loaded when the \texttt{\loadmath} is used. The macros here redefine globally all encoding dependent settings declared in the section 2.15.

\begin{verbatim}
\texttt{\_codedecl \_ncharrmA \{Uni math codes <2022-01-06>\} % preloaded on demand by \loadmath\texttt{\_chardef \_ncharbfA = "1D400 \_chardef \_ncharbfA = "1D41A}
\end{verbatim}

The control sequences for \texttt{\alpha}, \texttt{\beta} etc are redefined here. The \texttt{\alpha} expands to the character with Unicode \texttt{"03B1"}, this is a normal character \(\alpha\). You can type it directly in your editor if you know how to do this.

\begin{verbatim}
\texttt{\_chardef \_ncharma = "a \_chardef \_ncharma = "a}
\end{verbatim}

The math alphabets are declared here using the \texttt{\_umathrange\{range\}\{class\}\{family\}\{starting-code\} macro.

\begin{verbatim}
\texttt{\_protected \_def \_nvariables \{\_umathrange{A-Z}\text{\_ncharrmA \_umathrange{a-z}\text{\_ncharrma}}}
\end{verbatim}
Digits are configured like math alphabets.

The \texttt{\cal}, \texttt{\bbchar}, \texttt{\frak}, \texttt{\script} and the \texttt{\rm}, \texttt{\bf}, \texttt{\it}, \texttt{\bi}, \texttt{\tt} are defined here. Their “8bit definitions” from the file \texttt{math-preload.opm} (section 2.14) are removed. You can redefine them again if you need different behavior (for example you don’t want to use sans serif bold in math). What to do:

```latex
\protected\def\bf {\tryloadbf\tenbf \inmath{\bfvariables\bfgreek\bfGreek\bfdigits}}
\protected\def\bi {\tryloadbi\tenbi \inmath{\bivariables\bigreek\bfGreek\bfdigits}}
\public \bf \bi ;
```
\_inmath \{cmds\} applies \{cmds\} only in math mode.

\protected\def\inmath#1{\relax\ifmmode#1\relax\fi} % to keep off \loop processing in text mode
\% You can redefine these macros to follow your wishes.
\% For example, you need upright lowercase greek letters, you don't need
\% \bf and \bi behave as sans serif in math, ...
\protected\def\rm{\tryloadrm\tenrm\inmath{\rmvariables\rmdigits}}
\protected\def\it{\tryloadit\tenit\inmath{\itvariables\itGreek}}
\protected\def\bf{\tryloadbf\tenbf\inmath{\bsansvariables\bsansgreek\bsansGreek\bsansdigits}}
\protected\def\bi{\tryloadbi\tenbi\inmath{\bisansvariables\bisansgreek\bsansGreek\bsansdigits}}
\protected\def\tt{\tryloadtt\tentt\inmath{\ttvariables\ttdigits}}
\protected\def\bbchar{\bbvariables\bbdigits}
\protected\def\cal{\calvariables}
\protected\def\frak{\frakvariables}
\protected\def\misans{\isansvariables\sansdigits}
\protected\def\mbisans{\mbisansvariables\bisansgreek\bsansGreek\bsansdigits}
\protected\def\script{\scriptvariables\rmdigits\itgreek\rmGreek}
\public\rm\it\bf\bi\tt\bbchar\cal\frak\misans\mbisans\script\mit;

Each Unicode slot carries information about math type. This is saved in the file MathClass-15.txt which is copied to \texttt{mathclass.opm} The file has the following format:

\texttt{unimath-table.opm}

We have to read this information and convert it to the \texttt{\Umathcode}s.

Each math symbol has its declaration in the file \texttt{unicode-math-table.tex} which is copied to \texttt{unimath-table.opm}. The file has the following format:

\texttt{unimath-table.opm}
We have to read this information and convert it to the Unicode math codes.

Many special characters must be declared with care...

We have to read this information and convert it to the Unicode math codes.

Many special characters must be declared with care...
Aliases are declared here. They are names not mentioned in the unimath-table.opm file but commonly used in \TeX.
The \texttt{\not} macro is redefined here. If the \texttt{\not!(char)} is defined (by \texttt{\negationof}) then this macro is used. Else centered / is printed over the \langle char \rangle.
Newly declared public control sequences are used in internal macros by OpTEX. We need to get new meanings for these control sequences in the private namespace.

```latex
unimath-codes.opm
```

2.16.3 More Unicode-math examples

Example of using additional math font is in section 5.3 in the optex-math.pdf documentation

You can combine more Unicode math fonts in single formula simply by the \addUmathfont macro, see OpTEX trick 0030.


2.16.4 Printing all Unicode math slots in used math font

This file can be used for testing your Unicode-math font and/or for printing TeX sequences which can be used in math.

Load Unicode math font first (for example by \fontfam[termes] or by \loadmath{⟨math-font⟩}) and then you can do \input print-unimath.opm. The big table with all math symbols is printed.
2.17 Scaling fonts in document (high-level macros)

These macros are documented in section 1.3.2 from the user point of view.

\fontdef\small\setfontsize{at5pt} % sets text font size by the \setfontsize macro and math font sizes by setting internal macros \sizemtext, \sizemscript and \sizemsscript. It uses common concept font sizes: 100%, 70% and 50%. The \setmainvalues sets the parameters as main values when the \typsize is called first.

\txtypsize[\langle font-size\rangle/\langle baselineskip\rangle] sets given parameters. It sets text font size by the \setfontsize macro and math font sizes by setting internal macros \sizemtext, \sizemscript and \sizemsscript. It uses common concept font sizes: 100%, 70% and 50%. The \setmainvalues sets the parameters as main values when the \typsize is called first.
\protected\def \_typoscale [#1/#2]{{%  
  \ifs\&#1\&\def \_tmp{[/}\else  
  \_edef \_tmp{[\_ea\_ignorept \the\_tmpdim/]}\fi  
  \ifs\&#2\&edef \_tmp{[\_tmp]}\else  
  \_settmpdim{#2}\baselineskip  
  \_edef \_tmp{[\_tmp \_ea\_ignorept \the\_tmpdim]}\fi  
  \_ea\typosize \_tmp  
}  
\_def \_settmpdim#1#2{  
  \_tmpdim=#1pt \divide \_tmpdim by1000  
  \_tmpdim=\_ea\_ignorept \the\_tmpdim  
}  
\_public \_typoscale ;  
\_setbaselineskip \(\text{baselineskip}\) sets new \baselineskip and more values of registers which are dependent on the \(\text{baselineskip}\) including the \strutbox.
\_setmainvalues \{\text{baselineskip}\} sets new \baselineskip and more values of registers which are dependent on the \(\text{baselineskip}\) including the \strutbox.
\_setmainvalues \{\text{baselineskip}\} sets new \baselineskip and more values of registers which are dependent on the \(\text{baselineskip}\) including the \strutbox.
\_setmainvalues \{\text{baselineskip}\} sets new \baselineskip and more values of registers which are dependent on the \(\text{baselineskip}\) including the \strutbox.
\let \setmainvalues=\setmainvaluesL
\fi
\optsize=\mainfontsize \baselineskip=\mainbaselineskip
\}
\public \scalemain \mainfontsize \mainbaselineskip \mfontsrule ;

Suppose following example: \{\typosize[13/15] Let $M$ be a subset of $\mathbb{R}$ and $x \in M \ldots \}
If \mfontsrule=1 then \typosize does not load math fonts immediately but at the first math formula. It
is done by \everymath register, but the contents of this register is processed inside the math group.
If we do \everymath=\{\normalmath\} then this complicated macro will be processed three times in your
example above. We want only one processing, so we do \everymath=\{\setmathfonts\} and this macro
closes math mode first, loads fonts and opens math mode again.

\setmathfonts\{\normalmath\} \everymath{} \everydisplay{}

\thefontsize \{[size]\} and \thefontscale \{[factor]\} do modification of the size of the current font.
They are implemented by the \newcurrfontsize macro.

\em keeps the weight of the current variant and switches roman ↔ italic. It adds the italic correction
by the \additcorr and \afteritcorr macros. The second does not add italic correction if the next
character is dot or comma.

The \boldify macro does \let\rm\bf, \let\it\bi and \let\normalmath=\boldmath. All following
text will be in bold. If should be used after \typosize or \typoscale macros.
The internal \runboldmath macro runs \boldmath immediately if no delay of the math font loading
is set by \setmainvaluesL.

The \rm, \it in math mode must keep its original meaning.
We need to use a font selector for default pagination. Because we don't know what default font size will be selected by the user, we use this \_rmfixed macro. It sets the \rm font from the default font size (declared by first \typsize command and redefines itself be only the font switch for the next pages.

\begin{verbatim}
\def \rmfixed {%
\ifsdim \mainfosize=0pt
\mainfosize=10pt
\fi
\fontdef \tenrm {\setfontsize{at} \mainfosize} \resetmod \rm
\global \let \rmfixed = \tenrm % next use will be font switch only
\rmfixed
\let \rmfixed = \tenrm % user can redefine it
\end{verbatim}

2.18 Output routine

The output routine \optexoutput is similar as in plain \TeX. It does:

- \begoutput which does:
  - increments \pageno,
  - prints \Xpage{\langle \texttt{pageno} \rangle} to the .ref file (if \openref is active),
  - calculates \hoffset,
  - sets local meaning of macros used in headlines/footlines (see \regmacro).
- \shipout \completepage, which is \vbox of –
  - background box, if \pgbackground is non-empty,
  - headline box by \makeheadline, if the \headline is nonempty,
  - \vbox to\vsize of \pagecontents which consists of –
    - \pagedest, the page destination \texttt{pg:} \langle \texttt{pageno} \rangle for hyperlinks is created here,
    - \topins box if non-empty (from \topinserts),
    - \box255 with completed vertical material from main vertical mode,
    - \footnoterule and \footins box if nonempty (from \fnote, \footnote),
    - \pgbottomskip (default is 0 pt).
  - footline box by \makefootline, if the \footline is nonempty
- \endoutput which does:
  - increments \pageno using \advancepageno
  - runs output routine repeatedly if \dosupereject is activated.

\begin{verbatim}
\def \begoutput{
\incr \gpageno
\immediate \wref \Xpage{{\langle \texttt{\the \gpageno} \rangle}}{{\langle \texttt{\folio} \rangle}}
\sethsize \prepoffsets \the \regmark
\end{verbatim}
The \hsize value can be changed at various places in the document but we need to have a constant value \_xhsize in the output routine (for headlines and footlines, for instance). This value is set from the current value of \hsize when \_setxhsize macro is called. This macro destroys itself, so the value is set only once. Typically it is done in \margins macro or when first \optexoutput routine is called (see \begoutput). Or it is called at the beginning of the \begtt...\endtt environment before \hsize value is eventually changed by the user in this environment.

\_setxhsize \_global\_xhsize=\_hsize \_global\_let\_setxhsize=\_relax

\gpageno counts pages from one in the whole document

\_makeheadline creates \vbox to0pt with its contents (the \headline) shifted by \headlinedist up.

\_footnoterule is defined here.

\_pagecontents is similar as in plain \TeX. The only difference is that the \pagedest is inserted at the top of \pagecontents.

\_footnoterule is defined here.
Macros for footnotes are the same as in plain TeX. There is only one difference: \vfootnote is implemented as \opfootnote with empty parameter #1. This parameter should do local settings inside the \footins group and it does when \fnote macro is used. The \opfootnote nor \vfootnote don't take the footnote text as a parameter. This is due to a user can do catcode settings (like inline verbatim) in the footnote text. This idea is adapted from plain TeX. The \footnote and \footstrut is defined as in plain TeX.

The \topins macros \topinsert, \midinsert, \pageinsert, \endinsert are the same as in plain TeX.

The \draft macro is an example of usage \pgbackground to create watercolor marks.
2.19 Margins

The \margins macro is documented in the section 1.2.1.

The common page dimensions are defined here.
\magscale \[\langle factor \rangle\] does \texttt{\textbackslash mag=}\langle factor \rangle\) and recalculates page dimensions to their true values.

2.20 Colors

2.20.1 Basic concept

Setting of color in PDF is handled by graphics operators which change the graphics context. Colors for fills/strokes are distinguished, but apart from that, only one color is active at time and is used for all material drawn by following graphics operators, until next color is set. Each PDF content (e.g. page or form XObject) has its own graphics context, that is initialized from zero. Hence we have different concept of selecting fonts in \TeX{} (it depends on \TeX{} groups but does not depends on pages) and color handling in PDF.

\TeX{} itself has no concept of colors. Colors have always been handled by inserting \texttt{\textbackslash special} for DVI or using \texttt{\textbackslash pdf\textbackslash literal/\textbackslash pdf\textbackslash color\textbackslash stack} for PDF. It is very efficient and \TeX{} doesn’t even have to know anything about colors, but it is also problematic in many ways.

That is the reason why we decided to change color handling from \texttt{\textbackslash pdf\textbackslash color\textbackslash stack} to \LuaTeX{} attributes in version 1.04 of Op\TeX{}. Using attributes, the color setting behaves exactly like font selection from \TeX{} point of view: it respects \TeX{} groups, colors can span more pages, independent colors can be set for \texttt{\insert}s, etc. Moreover, once a material is created (using \texttt{\textbackslash setbox} for example) then it has its fonts and its colors frozen and you can rely on it when you are using e.g. \texttt{\textbackslash unhbox}. There are no internal whatsis for colors which can interfere with other typesetting material. In the end something like setting text to red (\texttt{\textcolor{Red}{text}}) should have the same nice behavior like setting text to bold (\texttt{\textbf{text}}).

\LuaTeX{} attributes can be set like count register – one attribute holds one number at time. But the value of attribute is propagated to each created typesetting element until the attribute is unset or set to another value. Very much like the font property. We use one attribute \texttt{\_colorattr} for storing the currently selected color (in number form).

Macros \texttt{\textbackslash set\_cmyk\textbackslash color\{}\langle C \rangle \langle M \rangle \langle Y \rangle \langle K \rangle\} or \texttt{\textbackslash set\_rgb\textbackslash color\{}\langle R \rangle \langle G \rangle \langle B \rangle\} or \texttt{\textbackslash set\_greycolor\{}\langle Grey \rangle\} are used in color selectors. These macros expand to internal \texttt{\_set\_color} macro which sets the \texttt{\_colorattr} attribute to an integer value and prepares mapping between this value and the real color data. This mapping is used just before each \texttt{\textbackslash shipout} in output routine. The \texttt{\_preshipout} pseudo-primitive is used here, it converts attribute values to internal PDF commands for selecting colors.

2.20.2 Color mixing

The color mixing processed by the \texttt{\textbackslash color\textbackslash def} is done in the subtractive color model CMYK. If the result has a component greater than 1 then all components are multiplied by a coefficient in order to the maximal component is equal to 1.

You can move a shared amount of CMY components (i.e. their minimum) to the \texttt{K} component. This saves the color tonners and the result is more true. This should be done by \texttt{\use\textbackslash K} command at the end of a linear combination used in \texttt{\color\textbackslash def}. For example

\texttt{\color\textbackslash def \myColor {.3\texttt{\textbackslash Green} + .4\texttt{\textbackslash Blue} \use\textbackslash K}}

The \texttt{\use\textbackslash K} command exactly does:

\[
k' = \min(C, M, Y),
\]
\[
C = (C - k')/(1 - k'), \quad M = (M - k')/(1 - k'), \quad Y = (Y - k')/(1 - k'), \quad K = \min(1, K + k').
\]

108
You can use minus instead of plus in the linear combination in \colordef. The given color is substracted in such case and the negative components are rounded to zero immediately. For example

\colordef \Color \Brown \Black

can be used for removing the black component from the color. You can use the −\Black trick after \useK command to remove grey components occurred during color mixing.

Finally, you can use “−” immediately preceeded before the macro name of the color. Then the complementary color is used here.

\colordef \mycolor \Grey .6 \Blue \% the same as \colordef \mycolor \Grey .6 \Yellow

The \rgbcolordef can be used to mix colors in additive color model RGB. If \onlyrgb is declared, then \colordef works as \rgbcolordef.

If a CMYK to RGB or RGB to CMYK conversion is needed then direct conversion of given color is used (if declared using \rgbcmykmap{⟨rgb⟩}{⟨cmyk⟩}) or the following simple formulae are used (ICC profiles are not supported):

\begin{align*}
\text{CMYK to RGB:} & \quad R = (1 - C)(1 - K), \quad G = (1 - M)(1 - K), \quad B = (1 - Y)(1 - K). \\
\text{RGB to CMYK:} & \quad K' = \max(R, G, B), \quad C = (K' - R)/K', \quad M = (K' - G)/K', \quad Y = (K' - B)/K', \quad K = 1 - K'.
\end{align*}

The RGB to CMYK conversion is invoked when a color is declared using \setrgbcolor and it is used in \colordef or if it is printed when \onlycmyk is declared. The CMYK to RGB conversion is invoked when a color is declared using \setcmykcolor and it is used in \rgbcolordef or if it is printed when \onlyrgb is declared.

### 2.20.3 Implementation

The basic colors in CMYK \Blue \Red \Brown \Green \Yellow \Cyan \Magenta \Grey \LightGrey \White and \Black are declared here.

\begin{verbatim}
\_def \Blue {\setcmykcolor{1 1 0 0}}
\_def \Red {\setcmykcolor{0 1 1 0}}
\_def \Brown {\setcmykcolor{0 .67 .67 .5}}
\_def \Green {\setcmykcolor{1 0 1 0}}
\_def \Yellow {\setcmykcolor{0 0 1 0}}
\_def \Cyan {\setcmykcolor{1 0 0 0}}
\_def \Magenta {\setcmykcolor{0 1 0 0}}
\_def \Grey {\setcmykcolor{0 0 0 .5}}
\_def \LightGrey {\setcmykcolor{0 0 0 .2}}
\_def \White {\setgreycolor{1}}
\_def \Black {\setgreycolor{0}}
\end{verbatim}

By default, the \setcmykcolor \setrgbcolor and \setgreycolor macros with \{\texttt{components}\} parameter expand to \_setcolor{\{\texttt{color-data}\}\{\texttt{fill-op}\}\{\texttt{stroke-op}\}} where \{\texttt{color-data}\} is \{R\} \{G\} \{B\} or \{C\} \{M\} \{Y\} \{K\} or \{G\} and \{\texttt{fill-op}\} is color operator for filling, \{\texttt{stroke-op}\} is color operator for stroking.

The \onlyrgb declaration redefines \setcmykcolor to do conversion to RGB just before \_setcolor is used. The \onlycmyk declaration redefines \setrgbcolor to do conversion to CMYK just before \_setcolor is used. Moreover, \onlyrgb re-defines three basic RGB colors for RGB color space and re-declares \colordef as \rgbcolordef.

109
The \_colorattr for coloring is allocated and \_setcolor{⟨color-data⟩}{⟨fill-op⟩}{⟨stroke-op⟩} is defined here. This macro does \_colorattr=\_colorcnt if the ⟨color data⟩ was not used before and prepare mapping from this integer value to the ⟨color data⟩ and increments _colorcnt. If the ⟨color data⟩ were used already, then \_setcolor does \_colorattr=⟨stored-value⟩. This work is done by the \_translatecolor macro. The following mapping macros are created:

\_color::⟨data⟩ {⟨fill-op⟩} ... expands to used ⟨attribute-value⟩
\_color:{⟨attribute-value⟩} ... expands to ⟨data⟩ {⟨fill-op⟩}
\_color-s:{⟨attribute-value⟩} ... expands to ⟨data⟩ {⟨stroke-op⟩}

The \_resetcolor un-sets the color attribute, it means that default color (black) shall be used.

We support concept of non-local color, i.e. all changes of the color attribute are global by setting \_colorprefix to \_global. \_localcolor is the default, i.e. \_colorprefix is \_relax. You can write \_global\Red if you want to have global setting of the color.

We use Lua codes for RGB to CMYK or CMYK to RGB conversions and for addition color components in the \_colordef macro. The \_rgbctocmyk ⟨R⟩ ⟨G⟩ ⟨B⟩ ; expands to ⟨C⟩ ⟨M⟩ ⟨Y⟩ ⟨K⟩ and the \_cmyktorgb ⟨C⟩ ⟨M⟩ ⟨Y⟩ ⟨K⟩ ; expands to ⟨R⟩ ⟨G⟩ ⟨B⟩. The \_colorcrop, \_colordefFin and \_douseK are auxiliary macros used in the \_colordef. The \_colorcrop rescales color components in order to they are in [0, 1] interval. The \_colordefFin expands to the values accumulated in Lua code color_C, color_M, color_Y and color_K. The \_douseK applies \useK to CMYK components. The \_tocmyk: ⟨rgb⟩ or \_torgb: ⟨cmyk⟩ control sequences (given by \_rgbctocmykmap) have precedence.
We have a problem with the %.3f directive in Lua code. It prints trailed zeros: (0.300 instead desired 0.3) but we want to save PDF file space. The macro \_stripzeros removes these trailing zeros at the expand processor level. So \_stripzeros 0.300 \_stripzeros 0.400 \_stripzeros 0.560 ; expands to \_stripzeros .3 \_stripzeros .4 \_stripzeros .56.

\rgbcolordef and \cmykcolordef use common macro \_commoncolordef with different first four parameters. The \_commoncolordef \_selectortype\_what\_define\_data does the real work. It initializes the Lua variables for summation. It expands \_data in the group where color selectors have special meaning, then it adjusts the resulting string by \replstring and runs it. Example shows how \_data are processed:

\_addcolor \_coefs\_mods\_types expands to \_addcolor\_mods\_types \_coefs for example it expands to \_addcolor\_K \_coefs followed by one or three or four numbers (depending on \_types). \_mods is = (use as is) or ~ (use complementary color). \_types is K for CMYK, R for RGB and G for GREY color space. Uppercase \_types informs that \cmykcolordef is processed and lowercase \_types informs that \rgbcolordef is processed. All variants of commands \_addcolor\_mod\_type\_coefs are defined. All of them expand to \_addcolor\_v1\_v2\_v3\_v4 which adds the values of Lua variables. The \rgbcolordef
uses \_addcolorA \{R\} \{G\} \{B\} 0 and \cmkykcolordef uses \_addcolorA \{C\} \{M\} \{Y\} \{K\}. So the Lua variable names are a little confusing when \rgbcolordef is processed.

Next, \_commoncolordef saves resulting values from Lua to \_tmpb using \_colordefFin. If \rgbcolordef is processed, then we must to remove the last \{K\} component which is in the format .0 in such case. The \_stripK macro does it. Finally, the \{what-define\} is defined as \{expanded _tmpb\}, for example \_setcmykclor\{1 0 .5 .3\}.

Public versions of \colordef and \useK macros are declared using \_def, because the internal versions \_colordef and \_useK are changed during processing.

The \LaTeX file x11nam.def is read by \morecolors. The numbers 0,1,2,3,4 are transformed to letters O, ⟨none⟩, B, C, D in the name of the color. Colors defined already are not re-defined. The empty \_showcolor macro should be re-defined for color catalog printing. For example:

\def\vr{\vrule height10pt depth2pt width20pt}
\def\_showcolor{\hbox{\tt\_bslash\_tmpb: \csname\_tmpb\endcsname \vr}}
colors.opm

\def\morecolors{%
\long\def\tmpb#1\replstring\tmpb(1){}\replstring\tmpb(2){}\%\ifcsname \tmpb\endcsname \else
\replstring\tmpb(3){}\replstring\tmpb(4){}\replstring\tmpb(5){}\fi\ea\tmpb
\fi}
\def\tmpa#1,#{2,3,4,5}{\ifx,#1,\else
\def\tmpb{#1}\replstring\tmpb{1}{}\replstring\tmpb{2}{B}\replstring\tmpb{3}{C}\replstring\tmpb{4}{D}\replstring\tmpb{0}{O}\fi\ifcsname \tmpb\endcsname \else
\sdef{\tmpb}{\setrgbcolor{#2 #3 #4}}\showcolor\fi\ea\tmpa\fi
\ea\tmp\input x11nam.def
\let\showcolor=\relax % re-define it if you want to print a color catalog
\public \morecolors ;

2.21 The .ref file

A so called .ref (jobname.ref) file is used to store data that will be needed in the next \TeX{} run (information about references, TOC lines, etc.). If it exists it is read by \everyjob, when processing of the document starts, but it is not created at all if the document doesn’t need any forward references. Here are the typical contents of a .ref file:

\Xrefversion{⟨ref-version⟩}
\_Xpage{⟨pageno⟩}{⟨pageno⟩}
\_Xtoc{⟨level⟩}{⟨type⟩}{⟨text⟩}{⟨title⟩}
\_Xlabel{⟨label⟩}{⟨text⟩}
\_Xpage{⟨pageno⟩}{langlepageno⟩}
\_Xlabel{⟨label⟩}{⟨text⟩}
...

- \_Xpage corresponds to the beginning of a page. ⟨pageno⟩ is an internal page number, globally numbered from one. ⟨pageno⟩ is the page number (⟨the\pageno⟩) used in pagination (they may differ).
- \_Xtoc corresponds to a chapter, section or subsection title on a page. ⟨title⟩ is the title of the chapter (⟨level⟩=1, ⟨type⟩=chap), section (⟨level⟩=2, ⟨type⟩=sec) or subsection (⟨level⟩=3, ⟨type⟩=secc).
- \_Xlabel corresponds to a labelled object on a page. ⟨label⟩ is the label provided by the user in \label{⟨label⟩}, while ⟨text⟩ is the text which should be used for the reference (section or table number, for example 2.3.14).

\_mdfive{⟨file⟩} expands to the MD5 hash of a given file. We use it to do consistency checking of the .ref file. First, we read the MD5 hash of .ref file from previous \TeX{} run before it is removed and opened for writing again in the \_inputref macro. The hash is saved to \_prevrefhash. Second, we
read the MD5 hash in the \_byehook macro again and if these hashes differ, warning that “ref file has changed” is printed. Try running \texttt{optex op-demo} twice to see the effect.

If the \texttt{.ref} file does not exist, then it is not created by default. This means that if you process a document without any forward references then no \texttt{\jobname.ref} file is created (it would be unusable). The \texttt{\_wref} macro is a dummy in that case.

If a macro needs to create and use the \texttt{.ref} file, then such macro must first use \texttt{\openref}. It creates the file and redefines \texttt{\_wref \{macro\} \{data\}} so that it saves the line \texttt{\{macro\}\{data\}} to the \texttt{.ref} file using the asynchronous \texttt{\write} primitive. Finally, \texttt{\openref} destroys itself, because we don’t need to open the file again.

\texttt{\_wref\{csname\}\{params\}} in fact does \texttt{\write\_reffile\{string\\{csname\}\{params\}\} and similarly \texttt{\_ewref\{csname\}\{params\}} does \texttt{\write\_reffile\{string\\{csname\}\{expanded-params\}\}.}

We are using the convention that the macros used in \texttt{.ref} file are named \texttt{\_X\langle foo\rangle}. We don’t want to read \texttt{.ref} files from old, incompatible versions of \TeX (and OPmac). This is ensured by using a version number and the \texttt{\Xrefversion} macro at the beginning of the \texttt{.ref} file:

\begin{verbatim}
\Xrefversion{<version>}
\end{verbatim}

The macro checks the version compatibility. Because OPmac does not understand \texttt{\_Xrefversion} we use \texttt{\Xrefversion} (with a different number of \texttt{\{version\}} than OPmac) here. The result: OPmac skips \texttt{.ref} files produced by \TeX and vice versa.

You cannot define your own \texttt{.ref} macros before \texttt{.ref} file is read because it is read in \texttt{\everyjob}. But you can define such macros by using \texttt{\refdecl{\{definitions of your ref macros\}}} This command immediately writes \texttt{\{definitions of your ref macros\}} to the \texttt{.ref} file. Then the next lines written to the \texttt{.ref} file can include your macros. An example from \texttt{CTUstyle2}:

\begin{verbatim}
\refdecl{"\%  \def\totlist{}  \def\toflist{}\^^J  \def\Xtab#1#2#3{\addto\totlist{\totline{#1}{#2}{#3}}}\^^J  \def\Xfig#1#2#3{\addto\toflist{\tofline{#1}{#2}{#3}}}\}
\end{verbatim}

We must read \texttt{\{definitions of your ref macros\}} while \# has the catcode 12, because we don’t want to duplicate each \# in the \texttt{.ref} file.
2.22 References

If the references are “forward” (i.e., the \ref is used first, the destination is created later) or if the reference text is page number then we must read .ref file first in order to get appropriate information. See section 2.21 for more information about .ref file concept.

\begin{verbatim}
\_def\textbf{Xpage}{\{gpageno\}\{pageno\}} saves the parameter pair into \textbf{\currcodepage}. Resets \textbf{\lfnotenum}; it is used if footnotes are numbered from one at each page.
\end{verbatim}

\begin{verbatim}
\_def\textbf{Xlabel}{\langle label\rangle}{\langle text\rangle} saves the \langle text\rangle to \textbf{\lab:\langle label\rangle} and saves \{\langle gpageno\rangle\}{\langle pageno\rangle} to \textbf{\pgref:\langle label\rangle}.
\end{verbatim}

\begin{verbatim}
\_def\textbf{ref}[#1]{\_xdef\_lastreflabel{#1}\_isnextchar\bgroup{\textbf{\refA}}{\textbf{\refA}\{@}}}
\_def\textbf{refA} #1{\_isdefined{\lab:\_lastreflabel}\_iftrue \_ilink[ref:\_lastreflabel]{\_reftext{\csname _lab:\_lastreflabel\_endcsname\}{#1}}\_else \_reftext{??}{#1}\_opwarning{label \_lastreflabel unknown. Try to TeX me again}\_incr\_unresolvedrefs \_openref\_fi}
\_def\textbf{pgref}[#1]{\_xdef\_lastreflabel{#1}\_isnextchar\bgroup{\textbf{\pgrefA}}{\textbf{\pgrefA}\{@}}}
\_def\textbf{pgrefA} #1{\_isdefined{\pgref:\_lastreflabel}\_iftrue \_ilink[pgref:\_lastreflabel]{\_reftext{\_csname _pgref:\_lastreflabel\_endcsname\}{#1}}\_else \_reftext{??}{#1}\_opwarning{label \_lastreflabel unknown. Try to TeX me again}\_incr\_unresolvedrefs \_openref\_fi}
\_def\textbf{\lastreflabel} keeps the value of the last label read by \textbf{\ref} or \textbf{\pgref}. You can use it for example by defining a macro \textbf{\pg} by \textbf{\def\pg}{\textbf{\pgref}\{\_lastreflabel\}} and then you need not repeat the same label in typical situations and you can write for instance: see section \textbf{\ref} at page \textbf{\pg}.
\end{verbatim}
2.23 Hyperlinks

There are six types of internal links and one type of external link used in \OpTeX. They are used in the format \((\text{type}):\langle\text{spec}\rangle\).

- \text{ref}:\langle\text{label}\rangle – the destination is created when \text{\label}\langle\text{label}\rangle is used, see also the section 2.22.
- \text{toc}:\langle\text{tocrefnum}\rangle – the destination is created at chap/sec/secc titles, see also the section 2.24.
- \text{pg}:\langle\text{gpageno}\rangle – the destination is created at beginning of each page, see also the section 2.18.
- \text{cite}:\langle\text{bibpart}\rangle/\langle\text{bibnum}\rangle – the destination is created in bibliography reference, see section 2.32.1.
- \text{fnt}:\langle\text{gfnotenum}\rangle – link form text to footnote, see also section 2.34.
- \text{fnf}:\langle\text{gfnotenum}\rangle – link from footnote to text, see also section 2.34.
- \text{url}:\langle\text{url}\rangle – used by \text{url} or \text{ulink}, see also the end of this section.

The \langle\text{tocrefnum}\rangle, \langle\text{gpageno}\rangle, \langle\text{bibnum}\rangle, and \langle\text{gfnotenum}\rangle are numbers starting from one and globally incremented by one in the whole document. The registers \text{\tocrefnum}, \text{\gpageno}, \text{\bibnum}, and \text{\gfnotenum} are used for these numbers.

When a chap/sec/secc title is prefixed by \text{\label}\langle\text{label}\rangle, then both types of internal links are created at the same destination place: \text{\toc}:\langle\text{tocrefnum}\rangle and \text{\ref}:\langle\text{label}\rangle.

The color for active links can be declared by \text{\def\langle\text{type}\rangle\text{linkcolor}}, the border around link can be declared by \text{\def\langle\text{type}\rangle\text{border}}. These macros are not declared by default, so color for active links are given only by \text{\hyperlinks} macro and borders are invisible. For example \text{\def\text{\toclinkcolor}{\Red}} means that links from table of contents are in red. Another example \text{\def\text{\tocborder}{1 0 0}} causes red frames in :\text{TOC} (not printed, only visible in PDF viewers).

\text{\dest}\langle\text{type}\rangle/\langle\text{spec}\rangle creates a destination of internal links. The destination is declared in the format \langle\text{type}\rangle/\langle\text{spec}\rangle. If the \text{\hyperlinks} command in not used, then \text{\dest} does nothing else it is set to \\text{\destactive}. The \text{\destactive} is implemented by \text{\pdfdest} primitive. It creates a box in which the destination is shifted by \text{\destheight}. The reason is that the destination is exactly at the top border of the PDF viewer but we want to see the line where the destination is. The destination box is positioned by a different way dependent on the current vertical or horizontal mode.
Each hyperlink is created internally by \_xlink{\langle type\rangle}{\langle spec\rangle}{\langle color\rangle}{\langle text\rangle}. This macro expands to \_quitvmode{\langle text\rangle} by default, i.e. no active hyperlink is created, only \langle text\rangle is printed in horizontal mode (and in a group). If \hyperlinks is used, then \_xlink gets the meaning of \_xlinkactive and hyperlinks are created by the \pdfstartlink/\pdfendlink primitives. The \langle text\rangle has given \langle color\rangle only when hyperlink is created. If \_linkcolor is defined, it has precedence over \langle color\rangle.

The \_linkdimens macro declares the dimensions of link area.

A specific action can be defined for each link \langle type\rangle by the macro \_action{\langle spec\rangle}. OpTEX defines only \urlaction{\langle url\rangle}. The default link action (when \_action{\langle spec\rangle} is not defined) is goto name{\langle type\rangle}{\langle spec\rangle} (an internal link). It is declared in the \linkactions{\langle type\rangle}{\langle spec\rangle} macro. The \pdfstartlink primitive uses attr{\_pdfborder{\langle type\rangle}}. The \pdfborder{\langle type\rangle} macro expands to \_urlborder{\langle type\rangle}{\langle spec\rangle} by default, i.e. \refborder, \citeborder, \tocborder, \pgborder, \urlborder, \fntborder or \fntfborder is defined.

\_link{\langle type\rangle}{\langle spec\rangle}{\langle color\rangle}{\langle text\rangle} creates a link. It is kept here for backward compatibility and it is equivalent to \_xlink{\langle type\rangle}{\langle spec\rangle}{\langle color\rangle}{\langle text\rangle}. If \_action{\langle spec\rangle} is not defined then \_link creates internal link do the \dest{\langle type\rangle}{\langle spec\rangle}. You can have more links with the same \langle type\rangle{\langle spec\rangle} but only one \dest in the document.

\_link{\langle type\rangle}{\langle spec\rangle}{\langle text\rangle} is equivalent to \_link but the \langle color\rangle is used from \hyperlinks declaration (or it is overwritten by \def\_linkcolor). \_link{\langle url\rangle}{\langle text\rangle} creates external link. The \langle url\rangle is detokenized with \_escapechar=-1 before it is used, so \%\%, \# etc. can be used in the \langle url\rangle.

\hyperlinks{\langle link color\rangle}{\langle ulink color\rangle} activates \_dest, \_xlink, so that they create links. Not setting colors {\hyperlinks{\{\}}} is also supported.

\url{\langle url\rangle} does approximately the same as \_ulink{\langle url\rangle}{\langle url\rangle}, but more work is done before the \url is processed. The link-version of \langle url\rangle is saved to \_tmpa and the printed version in \_tmpb. The printed version is processed in four steps: 1. the \_\% are replaced by [[1]] (we suppose that such string does
not exist in any URL). 2. it is detokenized with \escapechar=-1. 3. muti-strings and spaces are replaced by strings in braces \{ ... \}. 4. internal penalties and skips are put between characters using \_urlA, \_urlB and \_urlC. The step 4 do following: The \_urlskip is inserted between each pair of “normal characters”, i.e. characters not declared by \sdef{ur:⟨character⟩}. The special characters declared by \sdef{ur:⟨character⟩} are replaced by the body of their corresponding macro. The \_urlskip, \_urlbskip, \_urlgskip are typical skips used for special characters, their meaning is documented in the code below. You can change them. Default values: penalty 9990 is inserted between each pair of normal chararacters, penalty 100 is inserted after special charcters, nobreak before special characters. The URL can be broken at any place using these default values. If you want to disable breaking between normal characters, say \let\_urlskip\=nobreak.

The text version of the ⟨url⟩ is printed in \_urlfont.

\texttt{hyperlinks.om}'

\begin{verbatim}
123 \def\url#1{{% 134 \def\tmpa{#1}\replstring\tmpa \{\}|{}% 135 \def\tmpb{#1}\replstring\tmpb \{|{|}% 136 \replace\tmpb{[]}{{gb|}}% 137 \replace\tmpb{ }{{ }}% 138 \replace\tmpb{://}{{://}}% 139 \replace\tmpb{}% 140 \urlfont \urlA\tmpb\end}% 141 \def\urlA#1{\ifx\end#1\else \urlC{}{#1}\fi} 142 \def\urlB#1{\ifx\end#1\else \urlC{\urlxskip}{#1}\fi} 143 \def\urlC#1#2{% 144 \ifcsname _ur:#2\endcsname \lastnamedcs \else \urlA\urlB\urlC\end% 145 \fi} 146 \sdef{\url::://}{\urlskip:\urlskip:\urlskip/\urlbskip} 147 \sdef{\url:/}{\urlskip/\urlbskip} 148 \sdef{\url:.}{\urlskip.\urlbskip} 149 \sdef{\url:?}{\urlskip?\urlbskip} 150 \sdef{\url:=}{\urlskip=\urlbskip} 151 \sdef{\url:-}{\urlskip-\urlbskip} 152 \sdef{\url:&}{\urlskip\char`\&\urlbskip} 153 \sdef{\url:gb|}{\urlgskip} 154 \urlfont{\tt} % url font 155 \urlskip{(\penalty9990\hspace{0.1em}){\urlskip}\hspace{0.3em}\relax} % skip between normal characters 156 \urlskip{(\null\nobreak\hspace{0.1em}){\urlskip}\hspace{0.1em}\relax} % skip before \ // / . ? = - & 157 \urlskip{(\penalty100\hspace{0.1em}){\urlskip}\hspace{0.1em}\relax} % skip after \ // / . ? = - & 158 \urlskip{(\penalty-500\relax)} % "goodbreak" penalty generated by \| 159 \public \url ;
\end{verbatim}

\subsection*{2.24 Making table of contents}

\texttt{maketoc.om}'

\texttt{\\_Xtoc \{\langle level\rangle\}\{\langle type\rangle\}\{\langle number\rangle\}\{\langle o-title\rangle\}\{\langle title\rangle\}\{\langle gpageno\rangle\}\{\langle pageno\rangle\}}\ where:

\begin{itemize}
\item \langle level\rangle: 0 reserved, 1: chapter, 2: section, 3: subsection
\item \langle type\rangle: the type of the level, i.e. chap, sec, secc
\item \langle number\rangle: the number of the chapter/section/subsection in the format 1.2.3
\item \langle o-title\rangle: outlines title, if differs from \langle title\rangle.
\item \langle title\rangle: the title text
\item \langle gpageno\rangle: the page number numbered from 1 independently of pagination
\item \langle pageno\rangle: the page number used in the pagination
\end{itemize}

The last two parameters are restored from previous \texttt{\_Xpage\{\langle pageno\rangle\}\{\langle pageno\rangle\}}, data were saved in the \texttt{\_currpage} macro.

We read the \langle title\rangle parameter by \scantoeol from \texttt{.ref} file because the \langle title\rangle can include something like \{\}. 

118
\_def\_toclist{}\_newifi \_ifischap \_ischapfalse
\_def\_Xtoc#1#2#3#4{\_ifnum#1=0 \_ischaptrue\_fi
\_addto\_toclist{\_tocline{#1}{#2}{#3}{#4}}\_scantoeol \_XtocA}
\_def\_XtocA#1{\_addto\_toclist{{#1}}\_ea \_addto\_ea\_toclist\_ea{\_currpage}}
\_tocline{⟨level⟩}{⟨type⟩}{⟨number⟩}{⟨o-title⟩}{⟨title⟩}{⟨pageno⟩}{⟨pageno⟩} prints the record to the table of contents. It opens group, reduces \_leftskip, \_rightskip, runs the \_everytocline (user can customise the design of TOC here) and runs \_tocl:⟨level⟩{⟨number⟩}{⟨title⟩}{⟨pageno⟩} macro. This macro starts with vertical mode, inserts one record with given ⟨level⟩ and it should end by \_tocpar which returns to horizontal mode. The \_tocpar appends \_nobreak \_hskip-2\_iindent \_null \_par. This causes that the last line of the record is shifted outside the margin given by \_rightskip. A typical record (with long ⟨title⟩) looks like this:

\_llaptoclink⟨text⟩ does \_noindent \_llap{⟨linked text⟩}.
\_tocdotfill creates dots in the TOC.
\_nofirst\macro applies the \_macro only if we don’t print the first record of the TOC.
\_tocpar finalizes one TOC recors whith rapped ⟨pageno⟩.
\_pgn{⟨pageno⟩} creates ⟨pageno⟩ as link to real ⟨gpage⟩ saved in #6 of \_tocline. This is temporarily defined in the \_tocline.

The auxiliary macros are:

\_llaptoclink{⟨text⟩} does \_noindent \_llap{⟨linked text⟩}.
\_tocdotfill creates dots in the TOC.
\_nofirst\macro applies the \_macro only if we don’t print the first record of the TOC.
\_tocpar finalizes one TOC recors whith rapped ⟨pageno⟩.
\_pgn{⟨pageno⟩} creates ⟨pageno⟩ as link to real ⟨gpage⟩ saved in #6 of \_tocline. This is temporarily defined in the \_tocline.

If you want a special formatting of TOC with adding more special lines (no generated as titles from \chap, \sec, \secc), you can define \addtotoc{⟨level⟩}{⟨type⟩}{⟨number⟩}{⟨o-title⟩}{⟨title⟩} macro:
You can declare special lines (or something else) as an unused level (10 in the following example):
\sdef{\_tocl:10}{\blue-line}{}{\relax}{⟨blue text to be added in the TOC⟩}

Now, users can add a blue line into TOC by anywhere in the document. Note that \relax in the fourth parameter means that outline will be not generated. And second parameter blue-line is only a comment (unused in macros).

\maketoc prints warning if TOC data is empty, else it creates TOC by running \_toclist

\_newtoks \_regtoc \_newtoks \_regmark \_newtoks \_regoul
\def\_regmacro #1#2#3{% \toksapp\_regtoc(#1)\toksapp\_regmark(#2)\toksapp\_regoul(#3)%
}

\_public \maketoc \_regmacro ;

2.25 PDF outlines

2.25.1 Nesting PDF outlines

The problem is that PDF format needs to know the number of direct descendants of each outline if we need to create the tree of structured outlines. But we know only the level of each outline. The required data should be calculated from TOC data. We use two steps over TOC data saved in the \_toclist where each record is represented by one \_tocline.

The first step, the \_outlines macro sets \_tocline to \_outlinesA and calculates the number of direct descendants of each record. The second step, the \_outlines macro sets \_tocline to \_outlinesB and it uses prepared data and creates outlines.

Each outline is mapped to the control sequence of the type \_ol:⟨num⟩ or \_ol:⟨num⟩:⟨num⟩ or \_ol:⟨num⟩:⟨num⟩:⟨num⟩ etc. The first one is reserved for level 0, the second one for level 1 (chapters), the third one for level 2 (sections) etc. The number of direct descendants will be stored in these macros after the first step is finished. Each new outline of a given level increases the ⟨num⟩ at the given level. When the first step is processed then (above that) the \_ol:... sequence of the parent increases its value too. The \_ol:... sequences are implemented by \_ol:⟨count0⟩:⟨count1⟩:⟨count2⟩ etc. For example, when section (level 2) is processed in the first step then we do:

\_advance \count2 by 1
% increases the mapping pointer of the type
\_advance \_ol:\_count0:\_count1:\_count2 of this section
% increases the number of descendants connected
% to the parent of this section.

When the second step is processed, then we only read the stored data about the number of descendants. And we use it in count parameter of \_pdfoutline primitive.
For linking, we use the same links as in TOC, i.e. the toc:\_the\_tocrefnum labels are used.
\insertoutline \{\text\} inserts one outline with zero direct descendants. It creates a link destination of the type oul:\{num\} into the document (where \insertoutline is used) and the link itself is created too in the outline.

2.25.2 Strings in PDF outlines

There are only two encodings for PDF strings (used in PDFoutlines, PDFinfo, etc.). The first one is PDFDocEncoding which is single-byte encoding, but it misses most international characters.

The second encoding is Big Endian UTF-16 which is implemented in this file. It encodes a single character in either two or four bytes. This encoding is TeX-discomfortable because it looks like
This example shows a hexadecimal PDF string (enclosed in <> as opposed to the literal PDF string enclosed in ( )). In these strings each byte is represented by two hexadecimal characters (0-9, A-F). You can tell the encoding is UTF-16BE, because it starts with “Byte order mark” FEFF. Each unicode character is then encoded in one or two byte pairs. The example string corresponds to the text “Cvičení je zátěž a \( x \in \mathbb{M} \).” Notice the 4 bytes for the last character, \( \mathbb{M} \). (Even the whitespace would be OK in a PDF file, because it should be ignored by PDF viewers, but LuaTeX doesn’t allow it.)

\texttt{\_hexprint} is a command defined in Lua, that scans a number and expands to its UTF-16 Big Endian encoded form for use in PDF hexadecimal strings. The \texttt{\_hexprint} chunk to print hexadecimal numbers in the macro \texttt{\_hexprint}. Characters for quotes (and separators for quotes) are activated by first \texttt{\_scantextokens} and they are defined as the same non-active characters. But \texttt{\_regoul} can change this definition.
\def\removeoutbraces #1{\ifx#1\end\else\else\ea\removeoutbracesA\fi}
\def\removeoutbracesA #1{\ifx#1\end#1\else#1\ea\removeoutbraces\fi}
\def\removeoutmath #1$#2${\ifx#1\end#2\else#2\ea\removeoutmath\fi}

The \text{\textbackslash prepinverb}\langle\text{macro}\rangle\langle\text{separator}\rangle\{\langle\text{text}\rangle\}, e.g. \text{\textbackslash prepinverb}\tmpb|\{aaa |bbb| cccc |dd| ee\} does \texttt{\def\tmpb{|\langle\text{su}\rangle\{aaa }bbb\langle\text{su}\rangle\{ cccc }dd\langle\text{su}\rangle\{ ee\}}} where \langle\text{su}\rangle is \texttt{\scantextokens\unexpanded}. It means that in-line verbatim are not argument of \texttt{\scantextoken}. First \texttt{\edef\tmpb} tokenizes again the \langle\text{text}\rangle but not the parts which were in the the in-line verbatim.

\def\prepinverb#1#2#3{\def#1{}%\def\dotmpb ##1#2##2{\addto#1{\scantextokens\unexpanded{##1}}%\ifx\end##2\else\ea\dotmpbA\ea##2\fi}%\def\dotmpbA ##1#2{\addto#1{##1}\dotmpb}%\dotmpb#3#2\end}

The \texttt{\regmacro} is used in order to sed the values of macros \texttt{\em}, \texttt{\rm}, \texttt{\bf}, \texttt{\it}, \texttt{\bi}, \texttt{\tt}, \texttt{\slash} and \texttt{~} to values usable in PDF outlines.

\def\regmacro {}{}{\let\em=\empty \let\rm=\empty \let\bf=\empty \let\it=\empty \let\bi=\empty \let\tt=\empty \let\slash=\empty \let~=\space}

\public \pdfunidef;

2.26 Chapters, sections, subsections

We are using scaled fonts for titles \texttt{\titfont}, \texttt{\chapfont}, \texttt{\secfont} and \texttt{\seccfont}. They are scaled from main fonts size of the document, which is declared by first \texttt{\typosize[\langle\text{fo-size}\rangle/\langle\text{b-size}\rangle]} command.

\def\titfont {\scalemain\typoscale[\magstep4/\magstep5]\boldify}
\def\chapfont {\scalemain\typoscale[\magstep3/\magstep3]\boldify}
\def\secfont {\scalemain\typoscale[\magstep2/\magstep2]\boldify}
\def\seccfont {\scalemain\typoscale[\magstep1/\magstep1]\boldify}

The \texttt{\tit} macro is defined using \texttt{\scantoeol} and \texttt{\_printtit}. It means that the parameter is separated by end of line and inline verbatim is allowed. The same principle is used in the \texttt{\chap}, \texttt{\sec}, and \texttt{\secc} macros.

\def\_printtit #1{\vglue\titskip
\begin{center}
\let\leftskip=0pt plusfill \leftskip=\leftskip \% centering
\let\noindent=\noindent \scantextokens\unexpanded{#1}\par%
\nobreak\bigskip
\end{center}
\def\tit{\scantoeol\_printtit}

You can re-define \texttt{\_printchap}, \texttt{\_printsec} or \texttt{\_printsecc} macros if another design of section titles is needed. These macros get the \langle\text{title}\rangle text in its parameter. The common recommendations for these macros are:

- Use \texttt{\_above\{ penaltyA \}\{ \langle\text{penaltyA}\rangle \}} and \texttt{\_below\{ \langle\text{penaltyB}\rangle \}} for inserting vertical material above and below the section title. The arguments of these macros are normally used, i.e. \texttt{\_above\{ \langle\text{penaltyA}\rangle \}} inserts \langle\text{penaltyA}\rangle\langle\text{penaltyA}\rangle and \texttt{\_below\{ \langle\text{penaltyB}\rangle \}} inserts \langle\text{penaltyB}\rangle. But there is an exception: if \texttt{\_below\{ \langle\text{penaltyB}\rangle \}} is immediately followed by \texttt{\_above\{ \langle\text{penaltyA}\rangle \}\{ \langle\text{penaltyA}\rangle \}} (for example section title is immediately followed by subsection title), then only \langle\text{penaltyA}\rangle is generated, i.e. \langle\text{penaltyA}\rangle\langle\text{penaltyB}\rangle\langle\text{penaltyA}\rangle\langle\text{penaltyA}\rangle is reduced only to \langle\text{penaltyA}\rangle. The reason for such behavior: we don’t want to duplicate vertical skip and we don’t want to use the negative penalty in such cases. Moreover, \texttt{\_above\{ \langle\text{penaltyA}\rangle \}\{ \langle\text{penaltyB}\rangle \}} takes previous whatever vertical skip (other than from \texttt{\_below\{ \langle\text{penaltyB}\rangle \}} and generates only greater from this pair of skips. It means that \langle\text{whatever-skip}\rangle\langle\text{penaltyA}\rangle\langle\text{penaltyB}\rangle\langle\text{penaltyA}\rangle is transformed to \langle\text{penaltyA}\rangle\texttt{\max(\langle\text{whatever-skip}\rangle\langle\text{penaltyB}\rangle\langle\text{penaltyA}\rangle)}.
reason for such behavior: we don’t want to duplicate vertical skips (from \_belowlistskip, for example) above the title.

- Use \_printrefnum[⟨pre⟩⟨ref-num⟩⟨post⟩] in horizontal mode. It prints ⟨pre⟩⟨ref-num⟩⟨post⟩. The ⟨ref-num⟩ is \_thechapnum or \_thesecnum or \_theseccnum depending on what type of title is processed. If \nonum prefix is used then \_printrefnum prints nothing. The macro \_printrefnum does more work: it creates destination of hyperlinks (if \hyperlinks{}{} is used) and saves references from the label (if \label[⟨label⟩] precedes) and saves references for the table of contents (if \maketoc is used).
- Use \nbpar for closing the paragraph for printing title. This command inserts \_nobreak between each line of such paragraph, so the title cannot be broken into more pages.
- You can use \_firstnoindent in order to the first paragraph after the title is not indented.

\_def \_printchap #1\{\_vfill\_supereject
\_chapfont \_noindent \_mtext{chap} \_printrefnum[\@\]\_par
\_nobreak\_smallskip
\_noindent \_raggedright \#1\_nbpar\_mark{}%\}
\_nobreak \_belowtitle\(\_bigskip\%
\_firstnoindent
\}
\_def \_printsec#1\{\_par
\_abovetitle\(\_penalty-400\)_bigskip
\_secfont \_noindent \_raggedright \_printrefnum[\@\quad]#1\_nbpar\_insertmark\(\#1\)%
\_nobreak \_belowtitle\(\_medskip\%\)
\_firstnoindent
\}
\_def \_printsecc#1\{\_par
\_abovetitle\(\_penalty-200\)\_medskip\_smallskip
\_seccfont \_noindent \_raggedright \_printrefnum[\@\quad]#1\_nbpar\%
\_nobreak \_belowtitle\(\_medskip\%\)
\_firstnoindent
\}

The \_sectionlevel is the level of the printed section:

- \_sectionlevel=0 – reserved for parts of the book (unused by default)
- \_sectionlevel=1 – chapters (used in \chap)
- \_sectionlevel=2 – sections (used in \sec)
- \_sectionlevel=3 – subsections (used in \secc)
- \_sectionlevel=4 – subsubsections (unused by default, see the \LaTeX{} trick 0033)

\_newcount\_sectionlevel
\_def \_secinfo \(\_ifcase \_sectionlevel
\part\|\_or \chap\|\_or \sec\|\_or \secc\|\_or \seccc\_fi
\_globaldefs=1 \_chapx
\}

The \_chapx initializes counters used in chapters, the \_secx initializes counters in sections and \_seccx initializes counters in subsections. If you have more types of numbered objects in your document then you can declare appropriate counters and do \addto\_chapx\{\_yourcounter=0 \} for example. If you have another concept of numbering objects used in your document, you can re-define these macros. All settings here are global because it is used by \{\_globaldefs=1 \_chapx\}.

Default concept: Tables, figures, and display maths are numbered from one in each section – subsections don’t reset these counters. Footnotes declared by \fnotenumchapters are numbered in each chapter from one.

The \_the* macros \_thechapnum, \_thesecnum, \_theseccnum, \_thetnum, \_thefnum and \_thednum include the format of numbers used when the object is printing. If chapter is never used in the document then \_chapnum=0 and \_other\_chapnum. expands to empty. Sections have numbers ⟨num⟩ and subsections ⟨num⟩.⟨num⟩. On the other hand, if chapter is used in the document then \_chapnum=0 and sections have numbers ⟨num⟩.⟨num⟩ and subsections have numbers ⟨num⟩.⟨num⟩.⟨num⟩.
The `\notoc` and `\nonum` prefixes are implemented by internal `\ifnotoc` and `\ifnonum` macros. They are reset after each chapter/section/subsection by the `\resetnonumnotoc` macro.

The `\chap`, `\sec`, and `\secc` macros are implemented here. The `\inchap`, `\insec` and `\insecc` macros do the real work. First, we read the optional parameter `[⟨label⟩]`, if it exists. The `\chap`, `\sec` and `\secc` macro reads its parameter using `\scantoeol`. This causes that they cannot be used inside other macros. Use `\inchap`, `\insec`, and `\insecc` macros directly in such case.

The `\printrefnum[⟨pre⟩@⟨post⟩]` macro is used in `\print*` macros. Note that the `⟨title-text⟩` is `\detokenized` before `\wref`, so the problem of “fragile macros” from old \LaTeX never occurs. This fourth parameter is not delimited by `{...}` but by end of line. This gives possibility to have unbalanced braces in inline verbatim in titles.
\thisoutline{{\text}} saves text to the \_theoutline macro. \_printrefnum uses it and removes it.

\_def \_theoutline{}\_def \thisoutline#1{\_gdef \_theoutline{#1}}\_public \thisoutline ;

The \_abovetitle{\penalty\text} and \_belowtitle{\text} pair communicates using a special penalty 11333 in vertical mode. The \_belowtitle puts the vertical skip (its value is saved in \_savedtitleskip) followed by this special penalty. The \_abovetitle reads \_lastpenalty and if it has this special value then it removes the skip used before and doesn’t use the parameter. The \_abovetitle creates \_skipA only if whatever previous skip is less or equal than \_skipA. We must save \_whatever-skip, remove it, create \_penaltyA (if \_belowtitle does not precede) and create \_whatever-skip or \_skipA depending on what is greater. The amount of \_skipA is measured using \_setbox0=\vbox.

\_newskip \_savedtitleskip\_newskip \_savedlastskip\_def \_abovetitle #1#2{\_savedlastskip=\_lastskip \_ifdim \_lastskip>\_zo \_vskip=\_lastskip \_fi \_ifnum \_lastpenalty=11333 \_vskip=\_savedtitleskip \_else #1 \_fi \_ifdim \_savedlastskip>\_zo \_setbox0=\vbox{\_global \_tmpdim=\_lastskip} \_else \_tmpdim=\maxdimen \_fi \_ifdim \_savedlastskip>\_tmpdim \_vskip=\_savedlastskip \_else \_tmpdim \_fi \_def \_belowtitle #1{\_global \_savedtitleskip=\_lastskip \_penalty11333} \_nbpar sets \_interlinepenalty value. \_nl is “new line” in the text (or titles), but space in toc or headlines or outlines.

\_def \_nbpar{\_interlinepenalty=10000 \_endgraf} \_protected \_def \_nl{\_unskip \_hfil \_break} \_regmacro \_def \_nl{\_unskip \_space} \_regmacro \_def \_nl{\_unskip \_space} \_public \_nl; \_firstnoindent puts a material to \everypar in order to next paragraph will be without indentation. It is useful after titles. If you dislike this feature then you can say \_let \_firstnoindent=\_relax. The \_wipeepar removes the material from \everypar.

\_def \_firstnoindent{\_global \everypar={\_wipeepar \_setbox7=\_lastbox}} \_def \_wipeepar{\_global \everypar={}}

The \_mark (for running heads) is used in \_printsection only. We suppose that chapters will be printed after \vfil\break, so users can implement chapter titles for running headers directly by macros, no \_mark mechanism is needed. But sections need \_mark. And they can be mixed with chapter’s running heads, of course.

The \_insertmark{{\text \_title \text}} saves \_mark in the format {{\text-num \_title-text}}, so it can be printed “as is” in \_headline (see the space between them), or you can define a formatting macro with two parameters for processing these data, if you need it.

\_def \_insertmark#1{\_mark{{\_ifnonum \_else \_therefnum \_fi} \_unexpanded{#1}}} OpTeX sets \_headline=\text{} by default, so no running headings are printed. You can activate the running headings by following code, for example:

```
\addto\chapx {\edef \_runningchap {\_thechapnum: \_unexpanded \_ea{\_savedtitle}} \_def \_formathead #1\_2{\isempty{#1}\_iffalse #1: #2\_fi} \_headline = \% \_ifodd \_pageno \_hfil \_ea{\_formathead \_firstmark}{\%}
```
The `\secl{number} \{title-text\}` should be used for various levels of sections (for example, when converting from Markdown to \TeX). `\secl` is `\chap`, `\secl{2}` is `\sec`, `\secl{3}` is `\secc` and all more levels (for `{number}`> 3) are printed by the common `\seclp` macro. It declares only a simple design. If there is a requirement to use such more levels then the book designer can define something different here.

The `\caption/{letter}` increases `{letter}num` counter, edefines `{letter}num` as `{the}{letter}num` and defines `{the}caption` as language-dependent word using `{mtext}`, runs the `\everycaption{letter}` tokens register. The group opened by `\caption` is finalized by first `\par` from an empty line or from `\vskip` or from `\endinsert`. The `\cskip` macro inserts nonbreakable vertical space between the caption and the object.

The `\_printcaption{letter}` and `\_printcaptionf` macros start in vertical mode. They switch to horizontal mode and use `\wlabel{the}capnum` (in order to make reference and hyperlink destination) a they can use:

- `{the}caption` ... expands to the word Table or Figure (depending on the current language).
- `{the}num` ... expands to `{the}`num (caption number).

The `\_captionsep` inserts a separator between auto-generated caption number and the following caption text. Default separator is `\enspace` but if the caption text starts with dot or colon, then the space is not inserted. A user can write `\caption/t: My table` and "Table 1.1: My table" is printed. You can re-define the `\_captionsep` macro if you want to use another separator.

If you want to declare a new type of `\caption` with independent counter, you can use following lines, where `\_printcaption/a` for Algorithms are declared:

```latex
let \_printcaptiona = \_printcaptionf
let \_everycaptiona = \_everycaptionf
newcount \_anum
addto \sec {\_anum=0}
\def\_theanum {\_othelchapnum\_the\secnum\_the\_anum}
sdef\s{-mt:a:en}{Algorithm} sdef\s{-mt:a:cs}{Algorithmus} % + your language...
```
The default format of \caption text is a paragraph in block narrower by \_iindent and with the last line is centered. This setting is done by the \_narrowlastlinecentered macro.

\eqmark is processed in display mode (we add \eqno primitive) or in internal mode when \eqalignno is used (we don’t add \eqno).

The \numberedpar ⟨letter⟩ {⟨name⟩} is implemented here.

The \_printnumberedpar \theXnum {⟨name⟩} opens numbered paragraph and prints it. The optional parameter is in \_the_opt. You can re-define it if you need another design. \_printnumberedpar needs not to be re-defined if you only want to print Theorems in italic and to insert vertical skips (for example). You can do this by the following code:

\def\theorem {\medskip\bgroup\it \numberedpar A{Theorem}}
\def\endtheorem {\par\egroup\medskip}  
\theorem Let $M$ be... \endtheorem

\_setlistskip sets the skip dependent on the current level of items

2.27 Lists, items

\_code\begitems {Lists: begitems, enditems <2021-03-10>} % preloaded in format

\_aboveliskip is used above the list of items, \_belowliskip is used below the list of items and \_interliskip is used between items. \_listskipA is used as \_listskipamount at level 1 of items. \_listskipB is used as \_listskipamount at other levels. \_setlistskip sets the skip dependent on the current level of items
The \itemnum is locally reset to zero in each group declared by \begitems. So nested lists are numbered independently. Users can set initial value of \itemnum to another value after \begitems if they want. Each level of nested lists is indented by the new \iindent from left. The default item mark is \printitem.

The \begitems runs \aboveliskip only if we are not near below a title, where a vertical skip is placed already and where the \penalty 11333 is. It activates * and defines it as \startitem.

The \enditems runs \isnextchar \par \{ \noindent \} thus the next paragraph is without indentation if there is no empty line between the list and this paragraph (it is similar behavior as after display math).

\novspaces sets \listskipamount to 0pt.

Various item marks are saved in \item:\langle letter \rangle macros. You can re-define them or define more such macros. The \style \langle letter \rangle does \printitem={\item:\langle letter \rangle} first, then \style \langle letter \rangle does \printitem={\item:\langle letter \rangle} when it is used and finally, \startitem alias * uses \printitem.
\_athe\{⟨num⟩\} returns the ⟨num⟩s lowercase letter from the alphabet.
\_fullrectangle\{⟨dimen⟩\} prints full rectangle with given ⟨dimen⟩.

The \begblock macro selects fonts from footnotes \_fnset and opens new indentation in a group. \endblock closes the group. This is implemented as an counterpart of Markdown’s Blockquotes. Redefine these macros if you want to declare different design. The OpTeX trick 0031 shows how to create blocks with grey background splittable to more pages.

2.28 Verbatim, listings

2.28.1 Inline and “display” verbatim

\_setverb macro sets all catcodes to “verbatim mode”. It should be used only in a group, so we prepare a new catcode table with “verbatim” catcodes and we define it as \_catcodetable\_verbatimcatcodes. After the group is finished then original catcode table is restored.

\_defined\{\_code\{⟨text⟩\}\} expands to \detokenize\{⟨text⟩\} when \_escapechar=−1. In order to do it more robust when it is used in \write then it expands as noexpanded \_code\{⟨space⟩\} (followed by space in its csname). This macro does the real work.

The \_printinverbatim\{⟨text⟩\} macro is used for \_code\{⟨text⟩\} printing and for \`⟨text⟩\` printing.

When \_code occurs in PDF outlines then it does the same as \detokenize. The macro for preparing outlines sets \_escapechar to −1 and uses \_regoul token list before \edef.

The \_code is not \_protected because we want it expands to \unexpanded\{\_code\{⟨space⟩\}⟨text⟩\} in \write parameters. This protect the expansions of the \_code parameter (like \, \, etc.).
\verbchar\{char\} saves original catcode of previously declared \{char\} (if such character was declared) using \_savedttchar and \_savedttcharc values. Then new such values are stored. The declared character is activated by \_adef as a macro (active character) which opens a group, does \_setverb and other settings and reads its parameter until second the same character. This is done by the \_readverb macro. Finally, it prints scanned \{text\} by \_printinverbatim and closes group. Suppose that \verbchar" is used. Then the following work is schematically done:

\begin{verbatim}
\def "{\begingroup \_setverb ... \_readverb}
\def \_readverb #1"{\_printinverbatim{#1}\_endgroup}
\end{verbatim}

Note that the second occurrence of " is not active because \_setverb deactivates it.

\begtt
is defined only as public. We don’t need a private \begtt variant. This macro opens a group and sets \% as an active character (temporary). This will allow it to be used as the comment character at the same line after \begtt. Then \begtt is run. It is defined by \boldef, so users can put a parameter at the same line where \begtt is. This \#1 parameter is used after \everytt parameters settings, so users can change them locally.

The \begtt macro does \_setverb and another preprocessing, sets \endlinechar to \~\~J and reads the following text in verbatim mode until \endtt occurs. This scanning is done by \_startverb macro which is defined as:

\begin{verbatim}
\def\_startverb #1\endtt #2\~\~J{...}
\end{verbatim}

We must to ensure that the backslash in \endtt has category 12 (this is a reason of the \ea chain in real code). The \#2 is something between \endtt and the end of the same line and it is simply ignored.

The \_startverb puts the scanned data to \prepareverbdata. It sets the data to \_tmpb without changes by default, but you should re-define it in order to do special changes if you want. (For example, \hisyntax redefines this macro.) The scanned data have \~\~J at each end of line and all spaces are active characters (defined as \%). Other characters have normal category 11 or 12.

When \prepareverbdata finishes then \_startverb runs \_printverb loop over each line of the data and does a final work: last skip plus \noindent in the next paragraph.

The \_printverb macro calls \_printverbline\{\line\} repeatedly to each scanned line of verbatim text. The \_printverb is used from \begtt...\endtt and from \verbinput too.
The \_testcommentchars replaces the following \_iftrue to \_iffalse by default unless the \_commentchars are set. So, the main body of the loop is written in the \_else part of the \_iftrue condition. The \_printverbl\_\{line\}\_\} is called here.

The \_printverbl\_\{line\}\_\} expects that it starts in vertical mode and it must do \par to return the vertical mode. The \_printverbl\_\{line\}\_\} is used here: it does nothing when \_ttline<0 else it prints the line number using \_llap.

\_puttpenalty puts \_ttpenalty before second and next lines, but not before first line in each \begtt\_endtt environment.

```
162 \_def\_printverb #1^^J#2{%  
163 \_ifx\_printverblinenum\_relax \_else \_incr\_ttline \_fi  
164 \_iftrue  
165 \_ifx\_end#2 \_printcomments \_fi  
166 \_else  
167 \_ifx\_vcomments\_empty\_else \_printcomments \_def\_vcomments{}\_fi  
168 \_ifx\_end#2  
169 \_bgroup \_adef{ }{}\_def\_{}% if the last line is empty, we don't print it  
170 \_ifcat&#1&\_egroup \_else\_egroup \_printverbline{#1}\_fi  
171 \_else  
172 \_\_printverbline{#1}\%  
173 \_fi  
174 \_fi  
175 \_ifx\_end#2 \_let\_next=\_relax \_else \_def\_next{\_printverb#2}\_fi  
176 \_next  
177 \}  
178 \_def\_printverbl\_\{\_puttpenalty \_indent \_printverblinenum \_kern\_ttshift \#1\_\par\}  
179 \_def\_initverblinenum{\_tenrm \_thefontscale[700]\_ea\_let\_ea\_sevenrm \_the\_font}  
180 \_def\_printverblinenum{\_llap{\_sevenrm \_the\_ttline \_kern.9em}}  
181 \_def\_puttpenalty{\_def\_puttpenalty{\_penalty\_ttpenalty}}
```

Macro \verbinput uses a file read previously or opens the given file. Then it runs the parameter scanning by \_viscanparameter and \_viscanminus. Finally the \_doverbinput is run. At the beginning of \_doverbinput, we have \_viline= number of lines already read using previous \verbinput, \_vinolines = the number of lines we need to skip and \_vidolines = the number of lines we need to print. A similar preparation is done as in \begtt after the group is opened. Then we skip \_vinolines lines in a loop a and we read \_vidolines lines. The read data is accumulated into \_tmpb macro. The next steps are equal to the steps done in \_startverb macro: data are processed via \_prepareverbdata and printed via \_printverb loop.

```
198 \_def\_verbinput #1(#2) #3 {\_par \_def\_tmpa{#3}%  
199 \_def\_tmpb{#1}% cmds used in local group  
200 \_ifx\_vifilename\_tmpa \_else \_openin\_vifile={#3}%  
201 \_global\_viline=0 \_global\_let\_vifilename=\_tmpa  
202 \_ifeof\_vifile  
203 \_opwarning{\_string\verbinput: file "#3" unable to read}  
204 \_ea\_ea\_ea\_skiptorelax  
205 \_fi  
206 \fi  
207 \}  
208 \_def\_viscanparameter #2\_\relax  
209 \}  
210 \_def\_skiptorelax#1\_relax{}  
211 \_def \_viscanparameter #1+#2\_relax{%  
212 \_if$#2$ \_viscanminus(#1) \_else \_viscanplus(#1+#2) \_fi  
213 \}  
214 \_def \_viscanplus(#1+#2+){%  
215 \_if$#1$ \_tmpnum=\_viline  
216 \_else \_ifnum#1<0 \_tmpnum=\_viline \_advance\_tmpnum by-#1  
217 \_else \_tmpnum=#1  
218 \_fi  
219 \_fi  
220 \_edef\_vinolines{0}  
221 \_edef\_vidolines{#2}  
222 \_def \_vidolines(0) \_else \_edef \_vidolines\_\{#2\}\_\}  
223 \_doverbinput
```
\def\viscanminus(#1-#2){% 
  \_if$#1$\_tmpnum=0
  \else \_tmpnum=#1 \_advance\_tmpnum by-1 \_fi
  \_ifnum\_tmpnum<0 \_tmpnum=0 \_fi % (0-13) = (1-13)
  \_edef\vinolines{\_the\_tmpnum}%
  \_if$#2$\_tmpnum=0
  \else \_tmpnum=#2 \_advance\_tmpnum by-\_vinolines \_fi
  \_edef\vidolines{\_the\_tmpnum}%
  \_doverbinput}
\_def\doverbinput{%
  \_tmpnum=\_vinolines
  \_advance\_tmpnum by-\_viline
  \_ifnum\_ttline<-1 \_let\_printverblinenum=\_relax \_else \_initverblinenum \_fi
  \_setverb \_adef{ }{\_dsp} \_adef\^^I{\t} \_parindent=\_ttindent \_parskip=0pt
  \_def\t{\_hskip \_dimexpr\_tabspaces em/2\_relax}%
  \_protrudechars=0 % disable protrusion
  \_the\_everytt\_relax \_tmpb\_relax \_ttfont
  \_savemathsb \_endlinechar=`\^^J \_tmpnum=0
  \_loop \_ifnum\_ttline<\_viline \_ttline=\_the\_ttline
  \_edef\ttlinesave{\_ttline=\_the\_ttline}%
  \_ifnum\_ttline=-1 \_ttline=\_viline \_fi
  \_tmpnum=0 \_def\_tmpb{}%
  \_ifeof\_vifile \_tmpnum=\_vidolines\_space \_fi
  \_loop \_ifnum\_tmpnum<\_vidolines\_space
  \_vireadline
  \_ifnum\_vidolines=0 \_else\_advance\_tmpnum by1 \_fi
  \_ifeof\_vifile \_tmpnum=\_vidolines\_space \_else \_visaveline \_fi \_ttlinesave \_ttlinesave
  \_repeat
  \_ea\_prepareverbdata \_ea \_tmpb\_ea{\_tmpb^^J}%
  \_catcode`\_catcode`=&10 \_catcode`%=9 % used in \verbinput comments
  \_ea\_printverb \_tmpb\_end
  \_global\_ttlinesave
  \_par \_restoremathsb
  \_endgroup
  \_ttskip \_isnextchar\_par{}{\_noindent}%
  \}
\_define\vireadline{\_read\_vifile to \_tmp \_incr\_viline}
\_define\visaveline{\_ea\_addto\_ea\_tmpb\_ea{\_tmp}}
\_public \verbinput ;
\_savemathsb, \_restoremathsb pair is used in \begtt...\endtt or in \verbinput to temporary suppress the \mathsbon because we don’t need to print \int _a in verbatim mode if \int _a is really written. The \_restoremathsb is defined locally as \mathsbon only if it is needed.

\verbinput (4-) optex.lua here and the \commentchars -- was set before it.

If the language of your code printed by \verbinput supports the format of comments started by two characters from the beginning of the line then you can set these characters by \commentchars\_first\_second. Such comments are printed in the non-verbatim mode without these two characters and they look like the verbatim printing is interrupted at the places where such comments are. See the section 2.39 for good illustration. The file optex.lua is read by a single command \verbinput (4-) optex.lua here and the \commentchars -- was set before it.
If you need to set a special character by \commentchars then you must set the catcode to 12 (and space to 13). Examples:

\commentchars // % C++ comments
\commentchars -- % Lua comments
\commentchars \%=12 \_ea\commentchars %% % TeX comments
\commentchars \#=12 \catcode`\ =13 \_ea\commentchars#{ } % bash comments

There is one limitation when \texttt{TEX} interprets the comments declared by \commentchars. Each block of comments is accumulated to one line and then it is re-interpreted by \texttt{TEX}. So, the ends of lines in the comments block are lost. You cannot use macros which need to scan end of lines, for example \begtt...\endtt inside the comments block does not work. The character % is ignored in comments but you can use \% for printing or \% alone for de-activating \_endpar from empty comment lines.

Implementation: The \commentchars\(\text{first}\)\(\text{second}\) redefines the \_testcommentchars used in \_printverb in order to it removes the following \_iftrue and returns \_iftrue or \_iffalse depending on the fact that the comment characters are or aren’t present at the beginning of tested line. If it is true \(\text{\_ifnum 10}>0\) then the rest of the line is added to the \_vcomments macro.

The \_hicomments is \relax by default but it is redefined by \commentchars in order to keep no-colorized comments if we need to use feature from \commentchars.

The accumulated comments are printed whenever the non-comment line occurs. This is done by \_printcomments macro. You can re-define it, but the main idea must be kept: it is printed in the group, \_reloding \_rm initializes normal font, \catcodetable0 returns to normal catcode table used before \verbinput is started, and the text accumulated in \_vcomments must be printed by \_scantextokens primitive.

\verbatim.opm
\def\vcomments{}
\let\hicomments=\relax
\def\commentchars#1#2{%\def\testcommentchars ##1##2##3\relax ##4\_iftrue{\_ifnum % not closed in this macro
\_ifx #1##1\_ifx#2##21\fi\fi 0>0
\_ifx\relax##3\relax \_addto\_vcomments{\_endgraf}% empty comment=\enfgraf
\else \_addto\_vcomments{##3 }\fi}%
\def\hicomments{\_replfromto{\b
\#1#2}{\^\^J}{\w{#1#2####1}\^\^J}}% used in \hisyntax
\def\testcommentchars #1\_iftrue{\_iffalse} % default value of \_testcommentchar
\def\printcomments{\_ttskip
\catcodetable0 \_reloading \_rm \_everytt={\hisyntax{C}}%}
\_let\_vcomments=\relax% used in \_printcomments
\_ttskip
\_public \commentchars ;

The \visiblesp sets spaces as visible characters \(\text{ \_}.\) It redefines the \_dsp, so it is useful for verbatim modes only.

The \_dsp is equivalent to \_\ primitive. It is used in all verbatim environments: spaces are active and defined as \_dsp here.

\verbatim.opm
\def \_visiblesp{\_ifx\_initunifonts\relax \_def\_dsp{\_char9251 }%
\_else \_def\_dsp{\_char32 }\_fi}
\_let\_dsp=\ % primitive "direct space"
\_ttskip
\_public \visiblesp ;

2.28.2 Listings with syntax highlighting

The user can write

\begtt \hisyntax{C}
...
\endtt

to colorize the code using C syntax. The user can also write \everytt={\hisyntax{C}} to have all verbatim listings colorized.
\hisyntax{⟨name⟩} reads the file \hisyntax-{⟨name⟩}.opm where the colorization is declared. The parameter ⟨name⟩ is case insensitive and the file name must include it in lowercase letters. For example, the file \hisyntax-c.opm looks like this:

```
\_codedecl \_hisyntaxc {Syntax highlighting for C sources <2020-04-03>}
\_newtoks \_hisyntaxc \_newtoks \_hicolorsc
\_global\_hicolorsc={%
  colors for C language
  \_ hicolor K \Red % Keywords
  \_ hicolor S \Magenta % Strings
  \_ hicolor C \Green % Comments
  \_ hicolor N \Cyan % Numbers
  \_ hicolor P \Blue % Preprocessor
  \_ hicolor O \Blue % Non-letters
%
  \_edef \_tmpa {()
    \_string{\_string}+-*/=\[
  \_edef \_tmpa {^|!?}% non-letters
  \_ea \_foreach \_tmpa
    \_do {\_replthis{#1}{\n        \o#1\n}}
  \_foreach % keywords
    \_do {\_replthis{#1}{\z K{#1}}}
  \_replthis{.}{\n    .}
  \_replthis{e}{e}
  \_replthis{\e.}{.}
  \_replthis{\e.\c}{\c.}
  \_replthis{E\e.\c}{E-}
  \_edef \_o#1{\z O{#1}}
  \_edef \_c#1{\z N{#1}}
}
```

OpTeX provides \hisyntax-{c,python, tex, html}.opm files. You can take inspiration from these files and declare more languages.

Users can re-declare colors by \hicolors={...} This value has precedence over \_hicolors{⟨name⟩} values declared in the hicolors-{⟨name⟩}.opm file. The steps are: copy \_hicolors{⟨name⟩}={...} from hicolors-{⟨name⟩}.opm to your document, rename it to \hicolors={...} and do your own colors modifications.

Another way to set non-default colors is to declare \newtoks\hicolors{⟨name⟩} (without the _ prefix) and set the color palette there. It has precedence before \hicolors{⟨name⟩} (with the _ prefix) declared in the hicolors-{⟨name⟩}.opm file. This is useful when there are more hi-syntax languages used in one document.

Notes for hi-syntax macro writers
The file \hisyntax-{⟨name⟩}.opm is read only once and in a \TeX{} group. If there are definitions then they must be declared as global.

The file \hisyntax-{⟨name⟩}.opm must (globally) declare \_hisyntax{⟨name⟩} token list where the action over verbatim text is declared typically by using the \replfromto or \replthis macros.

The verbatim text is prepared by the pre-processing phase, then \_hisyntax{⟨name⟩} is applied and then the post-processing phase does final corrections. Finally, the verbatim text is printed line by line.

The pre-processing phase does:
• Each space is replaced by \n\n\n, so \n\textit{word}\n is the pattern for matching whole words (no subwords). The \n control sequence is removed in the post-processing phase.
• Each end of line is represented by \n\n\n.
• The \_start control sequence is added before the verbatim text and the \_end control sequence is appended to the end of the verbatim text. Both are removed in the post-processing phase.

Special macros are working only in a group when processing the verbatim text.
• \n represents nothing but it should be used as a boundary of words as mentioned above.
• \t represents a tabulator. It is prepared as \n\t\n because it can be at the boundary word boundary.
• \x \texttt{letter}\{\texttt{text}\} can be used as replacing text. Consider the example

\_replfromto{/*}{*/}{\texttt{/*#1*/}}

This replaces all C comments /... by \x \texttt{C{/*...*/}}. But C comments may span multiple lines, i.e. the --J should be inside it.

The macro \x \texttt{letter}\{\texttt{text}\} is replaced by one or more occurrences of \z \texttt{letter}\{\texttt{text}\} in the post-processing phase, each parameter \texttt{text} of \z is from a single line. Parameters not crossing line boundary are represented by \x \texttt{C{\texttt{text}}} and replaced by \z \texttt{C{\texttt{text}}} without any change. But:

\x \texttt{C{(text1)~~J(text2)~~J(text3)}}

is replaced by

\z \texttt{C{(text1)}}~~\z \texttt{C{(text2)}}~~\z \texttt{C{(text3)}}

\z \texttt{letter}\{\texttt{text}\} is expanded to \z: \texttt{letter}\{\texttt{text}\} and if \hicolor \texttt{letter} \texttt{color} is declared then \z: \texttt{letter}\{\texttt{text}\} expands to \texttt{color}\{\texttt{text}\}. So, required color is activated for each line separately (e.g. for C comments spanning multiple lines).
• \y \{\texttt{text}\} is replaced by \texttt{text} in the post-processing phase. It should be used for macros without a parameters. You cannot use unprotected macros as replacement text before the post-processing phase, because the post-processing phase is based on the expansion of the whole verbatim text.

The macros \texttt{replfromto} and \texttt{replthis} manipulate the verbatim text that is already stored in the \_tmpb macro.

\texttt{replfromto \{\texttt{from}\}\{\texttt{to}\}\{\texttt{replacement}\}} finds the first occurrence of \texttt{from} and the first occurrence of \texttt{to} following it. The \texttt{text} between them is packed into \#1 and available to \texttt{replacement} which ultimately replaces \texttt{text}.

\texttt{replfromto} continues by finding next \texttt{from}, then, next \texttt{to} repeatedly over the whole verbatim text. If the verbatim text ends with opening \texttt{from} but has no closing \texttt{to}, then \texttt{to} is appended to the verbatim text automatically and the last part of the verbatim text is replaced too.

The first two parameters are expanded before use of \texttt{replfromto}. You can use \texttt{csstring\%} or something else here.

The \texttt{replthis \{\texttt{pattern}\}\{\texttt{replacement}\}} replaces each \texttt{pattern} by \texttt{replacement}. Both parameters of \texttt{replthis} are expanded first.
The patterns ⟨from⟩, ⟨to⟩ and ⟨pattern⟩ are not found when they are hidden in braces {…}. E.g.

\replfromto{\{\}/*}{\}/*\{x \{C/\{}\}/*\} \nonumber

replaces all C comments by \x C{...}. The patterns inside {…} are not used by next usage of \replfromto or \replthis macros.

The \xscan macro replaces occurrences of \x by \z in the post-processing phase. The construct \x ⟨letter⟩{{{text}}} expands to \xscan ⟨letter⟩{{{text}}}”^J”. If \#3 is \end then it signals that something wrong happens, the ⟨from⟩ was not terminated by legal ⟨to⟩ when \replfromto did work. We must to fix this by using the \xscanR macro.

\def \xscanR#1\fi#2^^J \fi{\xscan{#1}#2^^J^} \nonumber

The \hicolor ⟨letter⟩ ⟨color⟩ defines \z: ⟨letter⟩{⟨text⟩} as {⟨color⟩⟨text⟩}. It should be used in the context of \x ⟨letter⟩{{{text}}} macros.

\def \hicolor #1#2{\sdef{\z:#1}{\cs{\z:}{#1}{#2}}} \nonumber

\hisyntax{⟨name⟩} re-defines default \prepareverbdata{⟨macro⟩}{⟨verbtext⟩}, but in order to do it does more things: It saves ⟨verbtext⟩ to \t, appends \n around spaces and ”^J” characters in pre-processing phase, opens hisyntax-{⟨name⟩}.opm file if \hisyntax{⟨name⟩} is not defined. Then \the\hisyntax{⟨name⟩} is processed. Finally, the post-processing phase is realized by setting appropriate values to the \x and \y macros and doing \edef \tmpb{}.

\def \tmpb{} \def \x{} \def \y{} \def \z{} \def \t{\hskip \dimexpr \tabspaces em/2}\relax

Aliases for languages can be declared like this. When \hisyntax{xml} is used then this is the same as \hisyntax{html}.
2.29 Graphics

The \texttt{\inspic} is defined by \texttt{\pdfximage} and \texttt{\pdfrefximage} primitives. If you want to use one picture more than once in your document, then the following code is recommended:

\begin{verbatim}
\newbox\mypic
\setbox\mypic = \hbox{\picw=3cm \inspic{⟨picture⟩}}
\end{verbatim}

My picture: \copy\mypic, again my picture: \copy\mypic, etc.

This code downloads the picture data to the PDF output only once (when \texttt{\setbox} is processed). Each usage of \texttt{\copy\mypic} puts only a pointer to the picture data in the PDF.

If you want to copy the same picture in different sizes, then choose a “basic size” used in \texttt{\setbox} and all different sizes can be realized by the \texttt{\transformbox{⟨transformation⟩}{\copy\mypic}}.

\texttt{\inspic} accepts old syntax \texttt{\inspic ⟨filename⟩⟨space⟩} or new syntax \texttt{\inspic{⟨filename⟩}}. So, we need to define two auxiliary macros \texttt{\_inspicA} and \texttt{\_inspicB}.

You can include more \texttt{\pdfximage} parameters (like \texttt{page(⟨number⟩)}) in the \texttt{\_picparams} macro.

All \texttt{\inspic} macros are surrounded in \texttt{\hbox} in order user can write \texttt{\moveright\inspic ...} or something similar.

Inkscape can save a picture to \texttt{*pdf} file and labels for the picture to \texttt{*pdf\_tex} file. The second file is in \LaTeX{} format (unfortunately) and it is intended to read immediately after \texttt{*pdf} is included in order to place labels of this picture in the same font as the document is printed. We need to read this \LaTeX{} file by plain \TeX{} macros when \texttt{\inkinspic} is used. These macros are stored in the \texttt{\_inkdefs} tokens list and it is used locally in the group. The solution is borrowed from OPmac trick 0032.
The \texttt{\transformbox{\{transformation\}}{\{text\}}} is copied from OPmac trick 0046. The \texttt{\rotbox{\{degrees\}}{\{text\}}} is a combination of \texttt{\rotsimple} from OPmac trick 0101 and the \texttt{\transformbox}. Note, that \texttt{\rotbox{-90}} puts the rotated text to the height of the outer box (depth is zero) because code from \texttt{\rotsimple} is processed. But \texttt{\rotbox{-90.0}} puts the rotated text to the depth of the outer box (height is zero) because \texttt{\transformbox} is processed.
\_scantwodimens scans two objects with the syntactic rule \langle \text{dimen} \rangle and returns \{\langle \text{number} \rangle\} \{\langle \text{number} \rangle\} in sp unit.

\puttext \langle \text{right} \rangle \langle \text{up} \rangle \{\langle \text{text} \rangle\} puts the \langle \text{text} \rangle to desired place: From current point moves \langle \text{down} \rangle and \langle \text{right} \rangle, puts the \langle \text{text} \rangle and returns back. The current point is unchanged after this macro ends.

\putpic \langle \text{right} \rangle \langle \text{up} \rangle \langle \text{width} \rangle \langle \text{height} \rangle \{\langle \text{image-file} \rangle\} does \puttext with the image scaled to desired \langle \text{width} \rangle and \langle \text{height} \rangle. If \langle \text{width} \rangle or \langle \text{height} \rangle is zero, natural dimension is used. The \nospec is a shortcut to such a natural dimension.

\backgroundpic \{\langle \text{image-file} \rangle\} puts the image to the background of each page. It is used in the \slides style, for example.

\circle{\langle x \rangle}{\langle y \rangle} creates an ellipse with \langle x \rangle axis and \langle y \rangle axis. The origin is in the center.

\oval{\langle x \rangle}{\langle y \rangle}{\langle roundness \rangle} creates an oval with \langle x \rangle, \langle y \rangle size and with the given \langle roundness \rangle. The real size is bigger by 2 \langle roundness \rangle. The origin is at the left bottom corner.

\mv{\langle x \rangle}{\langle y \rangle}{\langle curve \rangle} moves current point to \langle x \rangle, \langle y \rangle, creates the \langle curve \rangle and returns the current point back. All these macros are fully expandable and they can be used in the \pdfliteral argument.
The \texttt{\itoval{⟨text⟩}} is an example of \texttt{\_oval} usage.

The \texttt{\tincircle{⟨text⟩}} is an example of \texttt{\_circle} usage.

The \texttt{\ratio, \lwidth, \fcolor, \lcolor, \shadow \text{and} \overlapmargins} are parameters, they can be set by user in optional brackets […] For example \texttt{\fcolor=\Red} does \texttt{\_let\_fcolorvalue=\Red} and it means filling color.

The \texttt{\_setflcolors} uses the \texttt{\_setcolor} macro to separate filling (non-stroking) color and stroking color.

\begin{verbatim}
238    \newdimen \lwidth
239    \def \fcolor{\let \fcolorvalue}
240    \def \lcolor{\let \lcolorvalue}
241    \def \shadow{\let \shadowvalue}
242    \def \overlapmargins{\let \overlapmarginsvalue}
243    \def \ratio{\immediate \ratioA}{\ratioA=}
244    \def \ratioA #1 {\def \ratiovalue{#1}}
245    \def \touppervalue #1 {\ifx #1 N \let #1=N \fi}
246
247    \def \setflcolors #1 {% use only in a group
248      \edef #1{\fcolorvalue%}
249      \def \setcolor ##1 ##2 ##3 {##1 ##2} %
250      \edef #1 {#1 \space \lcolorvalue \space} %
251
252    \optdef \inoval {} \vbox \bgroup
253        \roundness=2pt \fcolor=\Yellow \lcolor=\Red \lwidth=.5bp
254        \shadow=N \overlapmargins=N \hhkern=Opt \vvkern=Opt
255        \the \ovalparams \relax \the \opt \relax
256        \touppervalue \overlapmarginsvalue \touppervalue \shadowvalue
257        \ifs \overlapmarginsvalue \N
258          \advance \hsize by-2 \hhkern \advance \hsize by-2 \roundness \fi
259      \setbox0=\hbox \bgroup \bgroup \aftergroup \inovalA \kern \hhkern \let \next=\%
260    \optdef \inovalA \egroup % of \setbox0=\hbox \bgroup
261        \ifdim \vvkern=\zo \else \ht0=\dimexpr \ht0+\vvkern \relax
262          \dp0=\dimexpr \dp0+\vvkern \relax \fi
263        \ifdim \hhkern=\zo \else \wd0=\dimexpr \wd0+\hhkern \relax \fi
264        \ifs \overlapmarginsvalue \N \dimen0=\roundness \dimen1=\roundness
265            \else
266              \dimen0=-\hhkern \dimen1=-\vvkern \fi
267        \setflcolors \tmp
268        \edef \tmpb {{\bp {\wd0+\lwidth}} {\bp {\ht0+\dp0+\lwidth}} {\bp {\roundness}}}
269        \doshadow \oval
270        \pdfliteral{q \bp {\lwidth} w \tmp \oval {\bp {\ht0+\dimen1} \relax \dp0=\dimexpr \dp0+\dimen1 \relax \space \box0 \kern \dimen0}}
271    \egroup % of \vbox \bgroup
272}\optdef \incircle {} \vbox \bgroup
273        \ratio=1 \fcolor=\Yellow \lcolor=\Red \lwidth=.5bp
274        \shadow=N \overlapmargins=N \hhkern=3pt \vvkern=3pt
275        \ea, \the \ea \circleparams \space \relax
276        \touppervalue \overlapmarginsvalue \touppervalue \shadowvalue
277      \setbox0=\hbox \bgroup \aftergroup \incircleA \kern \hhkern \let \next=\%
278    \optdef \incircleA \egroup % of \setbox0=\hbox \bgroup
279        \ht0=\dimexpr \ht0+\dimexpr \dp0+\vvkern \relax
280        \ifdim \ratiovalue {\dimexpr \ht0+\dp0 + \relax \dimexpr \ht0+\dp0} > \wd0
281          \dimen3=\dimexpr \ht0+\dp0 \dimen2=\ratiovalue \dimen3
282          \else \dimen2=\wd0 \dimen3=\exp \dimen2 \ratiovalue \dimen3 \fi
283    \setflcolors \tmp
284    \ifs \overlapmarginsvalue \N \dimen0=\zo \dimen1=\zo
285\end{verbatim}
Just before defining shadows, which require special graphics states, we define means for managing these graphics states. This is important, because otherwise our use of \pdfpageresources register might clash with other packages (TikZ) or even with our other usage (slides).

The macro \addextgstate⟨PDF name⟩ ⟨PDF dictionary⟩ shall be used for adding more graphics states. It must be used after \dump. First use of it detects PGF/TikZ and either uses its mechanism or defines our own. Our mechanism is very similar though – use single /ExtGState dictionary for all pages (\pdfpageresources just points to it).

A shadow effect is implemented here. The shadow is equal to the silhouette of the given path in a gray-transparent color shifted by \_shadowmoveto vector and with blurred boundary. A waistline with the width 2\*\_shadowb around the boundary is blurred. The \shadowlevels levels of transparent shapes is used for creating this effect. The \shadowlevels+1/2 level is equal to the shifted given path.

The \_doshadow{⟨curve⟩} does the shadow effect.
A generic macro \texttt{\_clipinpath\texttt{(x)}} \texttt{(y)}} \texttt{(curve)}} \texttt{(text)}} declares a clipping path by the \texttt{(curve)} shifted by the \texttt{(x)}, \texttt{(y)}. The \texttt{(text)} is typeset when such clipping path is active. Dimensions are given by \texttt{bp} without the unit here. The macros \texttt{\_clipinoval\texttt{(x)}} \texttt{(y)}} \texttt{(width)}} \texttt{(height)}} \texttt{(text)}} and \texttt{\_clipincircle\texttt{(x)}} \texttt{(y)}} \texttt{(width)}} \texttt{(height)}} \texttt{(text)}} are defined here. These macros read normal \TeX\ dimensions in their parameters.

\subsection{\texttt{\_table} macro, tables and rules}

\subsubsection{The \texttt{\_table} macro, \texttt{\_table}} macro, tables and rules}

The \texttt{(declaration)} part of \texttt{\_table\texttt{\{\_declaration\}}}\texttt{\{\_data\}} includes column declarators (letters) and other material: the \texttt{|} or \texttt{(\texttt{cmd})}. If the boundary declarator is not used then the boundaries of columns are just before each column declarator with exception of the first one. For example, the declaration \texttt{\{c|c:\(xx\)(yy)c\}} should be written more exactly using the boundary declarator by \texttt{\{c|c:\(xx\):(yy)c\}. But you can set these boundaries to other places using the boundary declarator explicitly, for example \texttt{\{c|c:\(xx\):(yy)c\}. The boundary declarator can be used only once between each pair of column declarators.
Each table item has its group. The \((cmd)\) are parts of the given table item (depending on the boundary declarator position). If you want to apply a special setting for a given column, you can do this by \((setting)\) followed by column declarator. But if the column is not first, you must use \((setting)\).

Example. We have three centered columns, the second one have to be in bold font and the third one have to be in red: \(\table{c: (\bf)c: (\Red)c}{\{data\}}\)

2.30.2 Usage of the \(\texttt{\textbackslash tabskip}\) primitive

The value of \texttt{\textbackslash tabskip} primitive is used between all columns of the table. It is glue-type, so it can be stretchable or shrinkable, see next section 2.30.3.

By default, \texttt{\textbackslash tabskip} is 0 pt. It means that only \texttt{\textbackslash tabiteml, \textbackslash tabitemr} and \((cmd)\) can generate visual spaces between columns. But they are not real spaces between columns because they are in fact the part of the total column width.

The \texttt{\textbackslash tabskip} value declared before the \texttt{\table} macro (or in \texttt{\everytable} or in \texttt{\thistable}) is used between all columns in the table. This value is equal to all spaces between columns. But you can set each such space individually if you use \texttt{\textbackslash tabskip=⟨value⟩} in the \(⟨\texttt{declaration}⟩\) immediately before boundary character. The boundary character represents the column pair for which the \texttt{\textbackslash tabskip} has individual value. For example \texttt{c(\textbackslash tabskip=5pt):r} gives \texttt{\textbackslash tabskip} value between \texttt{c} and \texttt{r} columns. You need not use boundary character explicitly, so \texttt{c(\textbackslash tabskip=5pt)r} gives the same result.

Space before the first column is given by the \texttt{\textbackslash tabskipl} and space after the last column is equal to \texttt{\textbackslash tabskipr}. Default values are 0 pt.

Use nonzero \texttt{\textbackslash tabskip} only in special applications. If \texttt{\textbackslash tabskip} is nonzero then horizontal lines generated by \texttt{\crli, \crlli and \crlp} have another behavior than you probably expected: they are interrupted in each \texttt{\textbackslash tabskip} space.

2.30.3 Tables to given width

There are two possibilities how to create tables to given width:

- \texttt{\table to⟨size⟩}{⟨declaration⟩}{⟨data⟩} uses stretchability or shrinkability of all spaces between columns generated by \texttt{\textbackslash tabskip} value and eventually by \texttt{\textbackslash tabskipl, \textbackslash tabskipr} values. See example below.
- \texttt{\table pxto⟨size⟩}{⟨declaration⟩}{⟨data⟩} expands the columns declared by \texttt{p⟨⟨size⟩⟩}, if the \(⟨\texttt{size}⟩\) is given by a virtual \texttt{\textbackslash tsize} unit. See the example below.

Example of \texttt{\table to⟨size⟩}:

\begin{verbatim}
\thistable{\textbackslash tabskip=0pt plus1fil minus1fil}
\table to\hsize {lr}{{\{data\}}}
\end{verbatim}

This table has its width \hsize. The first column starts at the left boundary of this table and it is justified left (to the boundary). The second column ends at the right boundary of the table and it is justified right (to the boundary). The space between them is stretchable and shrinkable to reach the given width \hsize.

Example of \texttt{\table pxto⟨size⟩} (means “paragraphs expanded to”):

\begin{verbatim}
\table pxto\hsize{\{\texttt{c|p⟨\tsize⟩}\}}{\{\texttt{crl}\}}
\hspace*{\textbackslash width}
\end{verbatim}

The first \(\texttt{c}\) column is variable width (it gets the width of the most wide item) and the resting space to given \hsize is filled by the \texttt{p} column.

You can declare more than one \texttt{p⟨⟨coefficient⟩\tsize⟩} columns in the table when \texttt{pxto} keyword is used.
This gives the ratio of widths of individual paragraphs in the table 3.5:2:1.

### 2.30.4 `\eqbox`: boxes with equal width across the whole document

The `\eqbox` \{\text{\label}\}\{\text{\texttt{\textit{text}}}\} behaves like `\hbox{\text{\texttt{\textit{text}}}}` in the first run of \TeX. But the widths of all boxes with the same label are saved to \texttt{.ref} file and the maximum box width for each label is calculated at the beginning of the next \TeX run. Then `\eqbox` \{\text{\label}\}\{\text{\texttt{\textit{text}}}\} behaves like `\hbox{\text{\texttt{\textit{text}}}\texttt{.\texttt{\label}}}`, where \texttt{\texttt{\textit{\dim:label}}} is the maximum width of all boxes labeled by the same \texttt{\texttt{\label}}. The documentation of the \LaTeX package \texttt{eqparbox} includes more information and tips.

The optional parameter \texttt{r} or \texttt{l} can be written before \texttt{\label}\{\text{\texttt{\textit{text}}}\} (for example `\eqbox \texttt{r}\{\text{\label}\}\{\text{\texttt{\textit{text}}}\}`) if you want to put the text to the right or to the left side of the box width.

Try the following example and watch what happens after first \TeX run and after the second one.

```latex
\def\leftitem#1{\par
  \noindent \hangindent=\eqboxsize\{\text{\texttt{\textit{items}}}\}2em\hangafter=1
  \eqbox r\{\text{\texttt{\textit{items}}}\}{#1 }\ignorespaces}
\leftitem \bf{first} \lorem[1]
\leftitem \bf{second one} \lorem[2]
\leftitem \bf{final} \lorem[3]
```

2.30.5 Implementation of the `\table` macro and friends

The result of the `\table\{\text{\texttt{\textit{declaration}}}\}\{\text{\texttt{\textit{data}}}}` macro is inserted into `\_tablebox`. You can change default value if you want by `\let\_tablebox=\vtop` or `\let\_tablebox=\relax`.

We save the \texttt{\texttt{\textit{to}}}\(\texttt{size}\) or \texttt{\texttt{\textit{pxt}}}\(\texttt{size}\) to \#1 and `\_tableW` sets the \texttt{\texttt{\textit{to}}}\(\texttt{size}\) to the `\_tablew` macro. If \texttt{\texttt{\textit{pxt}}}\(\texttt{size}\) is used then `\_tableW` is empty and `\_tmpdim` includes given \texttt{\texttt{\textit{size}}}\. The `\_ifpxto` returns true in this case.

The `\table` continues by reading \{\text{\texttt{\textit{declaration}}}\} in the `\_tableA` macro. Catcodes (for example the `\|` character) have to be normal when reading `\table` parameters. This is the reason why we use `\catcodetable` here.

The `\_tmptoks` is implemented by enlarging given `\_thistable` by desired dimension (height and depth too) and by setting \texttt{\_lineskip=-2\_\_tmptoks}. Normal table rows (where no `\_hrule` is between them) have normal baseline distance.

The `\_tableA\{\text{\texttt{\textit{declaration}}}\}` macro scans the \{\text{\texttt{\textit{declaration}}}\} by `\_scantabdata#1\relax` and continues by processing \{\text{\texttt{\textit{data}}}\} by `\_tableB`. The trick `\_tmptoks\{\text{\texttt{\textit{data}}}\}`, `\edef\_tmpb{\_the\_tmptoks}` is used here in order to keep the hash marks in the \{\texttt{\textit{data}}\} unchanged.
The \_tableB saves \langle data\rangle to \_tmpb and does \_tablereplstrings to prefix each macro \crl etc. by \crcr. See \_tablereplstrings. It cannot be used in a \table in another \table, so \_tablereplstrings is set to \relax locally.

The \_tabskip value is saved for places between columns into the \_tabskipmid macro. Then it runs

\_tablereplstrings replaces each \crl etc. to \crcr\crl. The reason is: we want to use macros that scan its parameter to a delimiter written in the right part of the table item declaration. The \crcr cannot be hidden in another macro in this case.

The \_scantabdata macro converts \table’s \langle declaration\rangle to \_halign \langle converted declaration\rangle. The result is stored into \_tabdata tokens list. For example, the following result is generated when \langle declaration\rangle=|cr||cl|.
The second result in the \_ddlinedata macro is a template of one row of the table used by \crli macro.

\begin{verbatim}
146 \_def\_scantabdata#1{\_let\_next=\_scantabdata
147 \_ifx\_relax#1\_let\_next=\_relax
148 \_else\_ifx#1\_def\_next{\_scantabdataE}
149 \_else\_ifx:#1\_def\_next{\_scantabdataF}
150 \_else\isinlist{123456789}#1\_iftrue \_def\_next{\_scantabdataC#1}
151 \_else \_ea\_ifx\_csname _tabdeclare#1\_endcsname \_relax
152 \_ea\_ifx\_csname _paramtabdeclare#1\_endcsname \_relax
153 \_opwarning{tab-declarator "#1" unknown, ignored}%
154 \_else
155 \_def\_next{\_ea\_scantabdataB\_csname _paramtabdeclare#1\_endcsname}
156 \_fi\_fi\_fi\_fi\_fi\_fi \\_next
157 }
158 \_def\_scantabdataA#1{\_addtabitem
159 \_ea\_addtabdata\_ea{#1\_tabstrutA \_tabskip\_tabskipmid\_relax}\_scantabdata}
160 \_def\_scantabdataB#1#2{\_addtabitem
161 \_ea\_addtabdata\_ea{#1\{#2\}_tabstrutA \_tabskip\_tabskipmid\_relax}\_scantabdata}
162 \_def\_scantabdataC {\_def\_tmpb{}\_afterassignment\_scantabdataD \_tmpnum=}
163 \_def\_scantabdataD#1{\_loop \_ifnum\_tmpnum>0 \_advance\_tmpnum by-1 \_addto\_tmpb{#1}\_repeat
164 \_ea\_scantabdata\_tmpb}
165 \_def\_scantabdataE#1){\_addtabdata{#1}\_scantabdata}
166 \_def\_scantabdataF {\_addtabitem\_def\_addtabitem{\_let\_addtabitem=\_addtabitemx}\_scantabdata}
\end{verbatim}

The \_addtabitemx adds the boundary code (used between columns) to the \_ddlinedata. This code is \_group \_& \_group \_colnum=value \_relax. You can get the current number of column from the \_colnum register, but you cannot write \_\the\_colnum as the first object in a \_data item because \_halign first expands the front of the item and the left part of the declaration is processed after this. Use \_relax\_the\_colnum instead. Or you can write:

\begin{verbatim}
\def\showcolnum{\_ea\_def\_ea\_totcolnum\_ea{\_the\_colnum}\\_the\_colnum/\_the\_totcolnum}
\table{ccc}{\_showcolnum & \_showcolnum & \_showcolnum}
\end{verbatim}

This example prints 1/3 2/3 3/3, because the value of the \_colnum is equal to the total number of columns before left part of the column declaration is processed.

\begin{verbatim}
\_newcount\_colnum % number of current column in the table
\_public \_colnum ;
\_def\_addtabitemx{\_ifnum\_colnum>0 \_addtabdata{&}\_addto\_ddlinedata{&\_dditem}_\_fi
181 \_advance\_colnum by1 \_let\_tmpa=\_relax
182 \_ifnum\_colnum>1 \_ea\_addtabdata\_ea{\_ea\_colnum\_the\_colnum\_relax}\_fi
183 \_ifnum\_colnum>0 \_addto\_vvleft{\_vvitem}\_else\_addto\_ddlinedata{\_vvitem}\_fi
184 \_ifnum\_colnum>1 \_ea\_addtabdata\_ea{\_ea\_colnum\_the\_colnum\_relax}\_fi
185 \_def\_addtabdata#1{\_tabdata\_ea{\_the\_tabdata#1}}
\end{verbatim}

This code converts \| or | from \table \_declaration to the \_ddlinedata. The default \_“declaration letters” c, l, r and p are declared by setting \_tabdeclarec, \_tabdeclarel, \_tabdeclarer and \_paramtabdeclarep macros. In general, define \_def\_tabdeclare(letter){...}
for a non-parametric letter and \def\_paramtabdeclare\{letter\}\{\ldots\} for a letter with a parameter. The double hash \#\# must be in the definition, it is replaced by a real table item data. You can declare more such “declaration letters” if you want.

Note, that the \#\# with fills are in group. The reason can be explained by following example:

\begin{verbatim}
\begin{table}[c!]{c|c|c!}
\hline
\text{Red A} & B
\hline
\end{table}
\end{verbatim}

We don’t want vertical line after red A to be in red.

The \_paramtabdeclarep\{\{data\}\} is invoked when \(p\{\{data\}\}\) declarator is used. First, it saves the \hspace value and then it runs \_tablepar. The \_tablepar macro behaves like \_tableparbox (which is \vtop) in normal cases. But there is a special case: if the first pass of pxto table is processed then \hspace is negative. We print nothing in this case, i.e. \_tableparbox is \ignoreit and we advance the \_tsizesum. The auxiliary macro \_tsizelast is used to do advancing only in the first row of the table. \_tsizesum and \_tsizelast are initialized in the \_tableB macro.

The \_tableparB initializes the paragraphs inside the table item and \_tableparC closes them. They are used in the \_paramtabdeclarep macro. The first paragraph is no indented.

Users put optional spaces around the table item typically, i.e. they write \& text & instead &text&. The left space is ignored by the internal \TeX algorithm but the right space must be removed by macros. This is a reason why we recommend to use \unskip after each \#\# in your definition of “declaration letters”. This macro isn’t only the primitive \unskip because we allow usage of plain \TeX \text\hidesskip macro: \&\hidesskip text\hidesskip&.

The \_tableparbox isn’t printed immediately, but \setbox2= is prefixed by the macro \_tableparA, which is empty by default (used in \_tablepar). The \_tableparB is processed after the box is set: it checks if there is only one line and prints \hbox to\hspace{\hfil\{this line\}\hfil} in this case. In other cases, the box2 is printed.

148
The family of \_cr* macros \crl, \crll, \crli, \crlli, \crlp and \tskip (\texttt{dimen}) is implemented here. The \_zerotabrule is used to suppress the negative lineskip declared by \tablinespace.

The \texttt{\mspan} \{(\texttt{number})\[\{(\texttt{declaration})\]\{\(\texttt{text}\}\} macro generates similar \\span\omit\span\omit\span sequence as plain \TeX macro \multispan. Moreover, it uses \_scantabdata to convert \{\texttt{declaration}\} from \texttt{table} syntax to \texttt{halign} syntax.

The \texttt{\vspan} \{(\texttt{number})\{\(\texttt{text}\)\} implementation is here. We need to lower the box by \{(\texttt{number})-1\}(\ht+\dp of \tabstrut) / 2.

The \#1 parameter must be a one-digit number. If you want to set more digits then use braces.

The parameters of primitive \texttt{\vrule} and \texttt{\hrule} keeps the rule “last wins”. If we re-define \texttt{\hrule} to \_orihrule height1pt then each usage of redefined \texttt{\hrule} uses 1pt height if this parameter isn’t
overwritten by another following `height` parameter. This principle is used for settings another default rule thickness than 0.4pt by the macro `\rulewidth`.

The `\frame{⟨text⟩}` uses "\vbox in \vtop" trick in order to keep the baseline of the internal text at the same level as outer baseline. User can write `\frame{abcxyz}` in normal paragraph line, for example and gets the expected result: `abcxyz`. The internal margins are set by `\vvkern` and `\hhkern` parameters.

\eqbox and \eqboxsize are implemented here. The widths of all \eqboxes are saved to the .ref file in the format `\_Xeqbox{⟨label⟩}{⟨size⟩}`. The .ref file is read again and maximum box width for each ⟨label⟩ is saved to `\_eqb:⟨label⟩`.

2.31 Balanced multi-columns

This code is documented in detail in the “\TeXbook naruby”, pages 244–246, free available, http://petr.olsak.net/tbn.html, but in Czech. Roughly speaking, macros complete all material between `\begmulti ⟨num-columns⟩` and `\endmulti` into one \vbox 6. Then the macro measures the amount of free space at the current page using `\pagegoal` and `\pagtotal` and does `\vsplit` of \vbox 6 to columns with a height of such free space. This is done only if we have enough amount of material in \vbox 6 to fill the full page by columns. This is repeated in a loop until we have less amount of material in \vbox 6. Then we run `\_balancecolumns` which balances the last part of the columns. Each part of printed material is distributed to the main vertical list as `\hbox{⟨columns⟩}` and we need not do any change in the output routine.

If you have paragraphs in `\begmulti...\endmulti` environment then you may say `\raggedright` inside this environment and you can re-assign `\widowpenalty` and `\clubpenalty` (they are set to 10000 in Op\TeX).

2.31 Balanced multi-columns
\_advance\_hsize by \_colsep
\_divide\_hsize by \_Ncols \_advance\_hsize by- \_colsep
\_mullines=0
\_def\_par{\_ifhmode\_endgraf\_global\_advance\_mullines by \_prevgraf\_fi}%
}\_def\_endmulti{\_vskip-\_prevdepth\_vfil
\_ifnum\_mullines<\_tmpnum \_dimen0=.8\_maxdimen \_else \_dimen0=.8\_maxdimen \_fi
\_divide\_dimen0 by \_Ncols \_relax
%% split the material to more pages?
\_ifvoid6 \_else
\_ifdim\_dimen0>\_dimen1 \_splitpart
\_else \_balancecolumns \_fi % only balancing
\_multiskip\_egroup
}\_def\_makecolumns{\_bgroup % full page, destination height: \dimen1
\_vbadness=20000 \_setbox1=\_hbox{}\_tmpnum=0
\_loop \_ifnum\_Ncols>\_tmpnum
\_advance\_tmpnum by1
\_setbox1=\_hbox{\_unhbox1 \_vsplit6 to\_dimen1 \_hss)
\_repeat
\_ifvoid6 \_else
\_ifdim\_dimen0>\_dimen1 \_else \_dimen0=0.8\_maxdimen \_fi
\_divide\_dimen0 by \_Ncols \_relax
\_ifx\_balancecolumns\_flushcolumns \_else \_dimen1=.5\_vsizem \_fi
\_ifdim\_dimen0=\_dimen2=\_dimen1
\_advance\_dimen2 by\_baselineskip
%% split the material to more pages?
\_ifvoid6 \_else
\_ifdim\_dimen0>\_dimen2 \_ea\_ea\_ea\_splitpart
\_else \_balancecolumns \_fi % last balancing
\_fi \_fi
}\_egroup
}\_def\_splitpart{\_makecolumns % full page
\_vskip0pt plus 1fil minus\_baselineskip \_break
\_ifnum\_mullines<\_tmpnum \_dimen0=.8\_maxdimen \_fi
\_divide\_dimen0 by \_Ncols \_relax
\_ifx\_balancecolumns\_flushcolumns \_else \_dimen1=.5\_vsizem \_fi
\_ifdim\_dimen0>\_dimen2 \_like\_tmp\_fi}
\_hbox{}\_nobreak\_vskip-\_splittopskip \_nointerlineskip
\_line{\_unhbox1\_unskip}
\_dimen0=\_dimen1 \_divide\_dimen0 by\_baselineskip \_multiply\_dimen0 by\_Ncols
\_global\_advance\_mullines by-\_dimen0
\_egroup
\_def\_balancecolumns{\_bgroup \_setbox7=\_copy6 % destination height: \dimen0
\_ifdim\_dimen0>\_baselineskip \_else \_dimen0=\_baselineskip \_fi
\_vbadness=20000
\_def\_tmp{\_setbox1=\_hbox{}\_tmpnum=0
\_loop \_ifnum\_Ncols>\_tmpnum
\_advance\_tmpnum by1
\_setbox1=\_hbox{\_unhbox1 \_vsplit6 to\_dimen1 \_hss)
\_repeat
\_ifvoid6 \_else
\_advance\_dimen0 by.2\_baselineskip
\_setbox6=\_copy7
\_close\_hfil\_vfil\_break
\_ifnum\_mullines<\_tmpnum \_dimen0=.8\_maxdimen \_else \_dimen0=0.8\_maxdimen \_fi
\_divide\_dimen0 by \_Ncols \_relax
\_ifx\_balancecolumns\_flushcolumns \_else \_dimen1=.5\_vsizem \_fi
\_ifdim\_dimen0>\_dimen2 \_like\_tmp\_fi}
\_hbox{}\_nobreak\_vskip-\_splittopskip \_nointerlineskip
\_line{\_unhbox1\_unskip}
we distinguish four cases:

2.32 Citations, bibliography

2.32.1 Macros for citations and bibliography preloaded in the format

Registers used by \cite, \bib macros are declared here. The \bibnum counts the bibliography items from one. The \bibmark is used when \nonumcitations is set.

\_bibp expands to \bibpart/. By default, \bibpart is empty, so internal links are in the form \cite:(\number). If \bibpart is set to \bibpart, then internal links are \cite:(\bibpart)/(\number).

\_citeA (label), \_nocite (label), \_rcite (label), \_ecite (\text) behaves like \cite but prints (\bib-marks) without brackets. The \text is hyperlinked like (\bib-marks) when \cite or \rcite is used. The empty internal macro \_savedcites will include the (\bib-marks) list to be printed. This list is set by \_citeA inside a group and it is used by \_printsavedcites in the same group. Each \cite/\rcite/\ecite macro starts from empty list of (\bib-marks) because new group is opened.

\_cite \_nocite \_rcite \_ecite \_printsavedcites.

\_ctlst: (\bibpart)/(\number) includes last number of bib-entry used in the document with given \bibpart. First one includes \_lastbn: \_ctlst: (\bibpart)/(\number) and second one includes \_lastbn: (\number). The \_lastbn: \_ctlst: macro includes last number of bib-entry used in the document with given \bibpart. A designer can use it to set appropriate indentation when printing the list of all bib-entries.

\_citeA (label), processes one label from the list of labels given in the parameter of \cite, \nocite, \rcite or \ecite macros. It adds the \label to a global list \_ctlst: \_ctlst: \_ctlst: which will be used by \usebib (it must know what \labels are used in the document to pick-up only relevant bib-entries from the database. Because we want to save space and to avoid duplications of \label in the \_ctlst: \_ctlst: , we distinguish four cases:
• ⟨label⟩ was not declared by \_\texttt{\_Xbib} before and it is first such a ⟨label⟩ in the document: Then \_\texttt{\_bib[(bibpart)]/⟨label⟩} is undefined and we save label using \_\texttt{\_addcitelist}, write warning on the terminal and define \_\texttt{\_bib[(bibpart)]/⟨label⟩} as empty.

• ⟨label⟩ was not declared by \_\texttt{\_Xbib} before but it was used previously in the document: Then \_\texttt{\_bib[(bibpart)]/⟨label⟩} is empty and we do nothing (only data to \_\texttt{\_savedcites} are saved).

• ⟨label⟩ was declared by \_\texttt{\_Xbib} before and it is first such ⟨label⟩ used in the document: Then \_\texttt{\_bib[(bibpart)]/⟨label⟩} includes \_\texttt{\_bibnn{⟨number⟩}}& and we test this case by the command \_\texttt{\_if \& \_\texttt{\_bibnn{⟨number⟩}}}\. This is true when \_\texttt{\_bibnn{⟨number⟩}} expands to empty. The ⟨label⟩ is saved by \_\texttt{\_addcitelist} and \_\texttt{\_bib[(bibpart)]/⟨label⟩} is re-defined directly as ⟨number⟩.

• ⟨label⟩ was declared by \_\texttt{\_Xbib} and it was used previously in the document. Then we do nothing (only data to \_\texttt{\_savedcites} are saved.

The \_\texttt{\_citeA} macro runs repeatedly over the whole list of ⟨labels⟩.

Because we implement possibility of more independent bibliography lists distinguished by ⟨bibpart⟩, the \_\texttt{\_addcitelist{⟨label⟩}} macro must add the ⟨label⟩ to given \_\texttt{\_ctlst:(⟨bibpart⟩/).}

When \_\texttt{\_addcitelist} is processed before \_\texttt{\_usebib}, then \_\texttt{\_citeI{⟨label⟩}} is added. \_\texttt{\_usebib} will use this list for selecting right records from .bib file. Then \_\texttt{\_usebib} sets \_\texttt{\_ctlst:(⟨bibpart⟩/} to \_\texttt{\_write}. If \_\texttt{\_addcitelist} is processed after \_\texttt{\_usebib}, then \_\texttt{\_Xcite{(⟨bibpart⟩/)}{⟨label⟩}} is saved to the .ref file. The \_\texttt{\_Xcite} creates \_\texttt{\_ctlstB:(⟨bibpart⟩/ as a list of saved \_\texttt{\_citeI{⟨label⟩}}. Finally, \_\texttt{\_usebib} concats both lists \_\texttt{\_ctlst:(⟨bibpart⟩/ and \_\texttt{\_ctlstB:(⟨bibpart⟩/ in the second \TeX run.

The ⟨bib-marks⟩ (in numeric or text form) are saved in \_\texttt{\_savedcites} macro separated by commas. The \_\texttt{\_printsavedcites} prints them by normal order or sorted if \_\texttt{\_sortcitations} is specified or condensed if \_\texttt{\_shordcitations} is specified.

The \_\texttt{\_sortcitations} appends the dummy number 300000 and we suppose that normal numbers of bib-entries are less than this constant. This constant is removed after the sorting algorithm. The \_\texttt{\_sortcitations} sets simply \_\texttt{\_lastcitenum}=1. The macros for ⟨bib-marks⟩ printing follows (sorry, without detail documentation). They are documented in opmac-d.pdf (but only in Czech).
The \texttt{\textcolor{cite-bib-opm}{\texttt{\textbackslash bib} \{\textcolor{cite-bib-opm}{\texttt{label}}\}}} or \texttt{\textcolor{cite-bib-opm}{\texttt{\textbackslash bib} \{\textcolor{cite-bib-opm}{\texttt{label}}\} =\{\texttt{bib-mark}\}}} prints one bib-entry without reading any database. The bib-entry follows after this command. This command counts the used \texttt{\textcolor{cite-bib-opm}{\texttt{\textbackslash bibnum}}} counter and saves \texttt{\textcolor{cite-bib-opm}{\texttt{\_Xbib}\{\textcolor{cite-bib-opm}{\texttt{bibpart}}\}{\textcolor{cite-bib-opm}{\texttt{label}}}{\textcolor{cite-bib-opm}{\texttt{number}}} {\textcolor{cite-bib-opm}{\texttt{nonumber}}}}} into \texttt{\textcolor{cite-bib-opm}{\texttt{\_wbib}\{\textcolor{cite-bib-opm}{\texttt{label}}\}{\textcolor{cite-bib-opm}{\texttt{number}}} {\textcolor{cite-bib-opm}{\texttt{nonumber}}}}} immediately using \texttt{\textcolor{cite-bib-opm}{\texttt{\_vbib}\{\textcolor{cite-bib-opm}{\texttt{label}}\}{\textcolor{cite-bib-opm}{\texttt{number}}} {\textcolor{cite-bib-opm}{\texttt{nonumber}}}}} . This is the core of creation of mapping from \texttt{\textcolor{cite-bib-opm}{\texttt{\{labels\}}} to \texttt{\textcolor{cite-bib-opm}{\texttt{\{number\}}} and \texttt{\textcolor{cite-bib-opm}{\texttt{\{nonumber\}}} . \texttt{\textcolor{cite-bib-opm}{\texttt{\_bibA}}} and \texttt{\textcolor{cite-bib-opm}{\texttt{\_bibB}}} implement the scanner of the optional argument with the \texttt{\textcolor{cite-bib-opm}{\texttt{\_bibmark}}} . \texttt{\textcolor{cite-bib-opm}{\texttt{\_bibgl}}} is \texttt{\textcolor{cite-bib-opm}{\texttt{\texttt{\_relax}}} by default but \texttt{\textcolor{cite-bib-opm}{\texttt{\_slides}}} do \texttt{\textcolor{cite-bib-opm}{\texttt{\_let\_bibgl=\_global}}} . \texttt{\textcolor{cite-bib-opm}{\texttt{\_dbib}}} creates destination for hyperlinks.
The \_printbib prints the bib-entry itself. You can re-define it if you want a different design. The \_printbib starts in horizontal mode after \noindent and after the eventual hyperlink destination is inserted. By default, the \_printbib sets the indentation by \hangindent and prints numeric \llap{⟨bib-marks⟩} by \llap{⟨the\bibnum⟩}. If \nonumcitations then the \_citelinkA is not empty and \langle bib-marks⟩ are not printed. The text of bib-entry follows. User can create this text manually using \bib command or it is generated automatically from a .bib database by \usebib command.

The vertical space between bib-entries is controlled by \_bibskip macro.

\usebib command is implemented in usebib.opm file which is loaded when the \usebib command is used first. The usebib.opm file loads the librarian.tex for scanning the .bib files. See the section 2.32.2, where the file usebib.opm is documented.

\nobibwarning [[list of bib-labels]] declares a list of bib labels which are not fully declared in .bib file but we want to suppress the warning about it. List of bib labels are comma-separated case sensitive list without spaces.

2.32.2 The \usebib command

The file usebib.opm implements the command \usebib/⟨sorttype⟩ ⟨⟨style⟩⟩ ⟨⟨bibfiles⟩⟩ where ⟨sorttype⟩ is one letter c (references ordered by citation order in the text) or s (references ordered by key in the style file), ⟨⟨style⟩⟩ is the part of the name bib-⟨⟨style⟩⟩.opm of the style file and ⟨⟨bibfiles⟩⟩ are one or more .bib file names without suffix separated by comma without space. Example:

\usebib/s (simple) mybase,yourbase

This command reads the ⟨⟨bibfiles⟩⟩ directly and creates the list of bibliographic references (only those declared by \cite or \nocite in the text). The formatting of such references is defined in the style file.

The principle “first entry wins” is used. Suppose \usebib/s (simple) local,global. If an entry with the same label is declared in local.bib and in global.bib too then the first wins. So, you can set exceptions in your local.bib file for your document.

The bib-⟨⟨style⟩⟩.opm declares entry types (like @BOOK, @ARTICLE) and declares their mandatory and optional fields (like author, title). When a mandatory field is missing in an entry in the .bib file then a warning is printed on the terminal about it. You can suppress such warnings by command \nobibwarning [[⟨bib-labels⟩]], where ⟨⟨bib-labels⟩⟩ is a comma-separated list of labels (without spaces) where missing mandatory fields will be no warned.

Old .bib files may use the obscure notation for accents like \'{o}. Recommendation: convert such old files to Unicode encoding. If you are unable to do this then you can set \bibtexhook={\oldaccents}.
2.32.3 Notes for bib-style writers

The .bib files include records in the format:

@{entry-type}{{\{label\},
  \{field-name\} = "\{field-data\}"
  \{field-name\} = "\{field-data\}"
  ...
  }
}

see the file demo/op-biblist.bib for a real example. The \{entry-types\} and \{field-names\} are case insensitive.

Ancient Bib\TeX\ has read such files and has generated files appropriate for reading by L\TeX. It has worked with a set of \{entry-types\}, see the www page \texttt{http://en.wikipedia.org/wiki/BibTeX}. The set of entry types listed on this www page is de facto the Bib\TeX\ standard. The Op\TeX\ bib style writer must “declare” all such entry types and more non-standard entry types can be declared too if there is a good reason for doing it. The word “declare” used in the previous sentence means that a bib-style writer must define the printing rules for each \{entry-type\}. The printing rules for \{entry-type\} include: which fields will be printed, in what order, by what format they will be printed on (italic, caps, etc.), which fields are mandatory, which are optional, and which are ignored in .bib records.

The style writer can be inspired by two styles already done: bib-simple.opm and bib-iso690.opm. The second one is documented in detail in section 2.32.5.

The printing rules for each \{entry-type\} must be declared by \_\sdef{\_print:\{entry-type\}} in bib-\{style\}.opm file. The \{entry-type\}-type has to be lowercase here. Op\TeX supports following macros for a more comfortable setting of printing rules:

- \_bprinta \{field-name\} \{if defined\} \{if not defined\}. The part \{if defined\} is executed if \{field-name\} is declared in .bib file for the entry which is currently processed. Else the part \{if not defined\} is processed. The part \{if defined\} can include the * parameter which is replaced by the value of the \{field-name\}.
- The part \{if not defined\} can include the \_bibwarning command if the \{field-name\} is mandatory.
- \_bprintb \{field-name\} \{if defined\} \{if not defined\}. The same as \_bprinta, but the #1 parameter is used instead *. Differences: #1 parameter can be used more than once and can be enclosed in nested braces. The * parameter can be used at most once and cannot be enclosed in braces. Warning: if the \_bprintb commands are nested (\_bprintb in \_bprintb), then you need to write the ####1 parameter for internal \_bprintb. But if \_bprinta commands are nested then the parameter is not duplicated.
- \_bprintc \macro \{if non-empty\}. The \{if non-empty\} part is executed if \macro is non-empty. The * parameter can be used, it is replaced by the \macro.
- \_bprintv \{field1,field2,...\} \{if defined\} \{if not defined\}. The part \{if defined\} is executed if \{field1\} or \{field2\} or ... is defined, else the second part \{if not defined\} is executed. There is one filed name or the list field names separated by commas. The parts cannot include any parameters.

There are two special field-names: !author and !editor. The processed list of authors or editors are printed here instead of raw data, see the commands \_authorname and \_editorname below.

The bib-style writer can define _\print:BEGIN and/or _\print:END. They are executed at the beginning or end of each \{entry-type\}. The formatting does not solve the numbering and paragraph indentation of the entry. This is processed by \_printbib macro used in Op\TeX\ (and may be redefined by the author or document designer).

The \texttt{\bibmark=\{something\}} can be declared, for instance in the \_print:END macro. Such “bibmark” is saved to the .ref file and used in next \TeX\ run as \cite marks when \nonumcitations is set.

Moreover, the bib-style writer must declare the format of special fields author and editor. These fields include a list of names, each name is processed individually in a loop. The \_authorname or \_editorname is called for each name on the list. The bib-style writer must define the \_authorname and \_editorname commands in order to declare the format of printing each individual name. The following control sequences can be used in these macros:

- \_NameCount: the number of the currently processed author in the list
- \_namecont: the total number of the authors in the list
- \_Lastname, \_Firstname, \_Von, \_Junior: the parts of the name.
The whole style file is read in the group during the `\usebib` command is executed before typesetting the reference list. Each definition or setting is local here.

The auto-generated phrases (dependent on current language) can be used in bib-style files by `\_mtext{bib.<identifier>}`, where `<identifier>` is an identifier of the phrase and the phrase itself is defined by `\_sdef{mt:bib.<identifier>}{language}{phrase}`. See section 2.37.3 for more detail. Phrases for `<identifiers>`: and, etal, edition, citedate, volume, number, prepages, postpages, editor, editors, available, availablealso, bachtise, matsthsis, phdthesis are defined already, see the end of section 2.37.3.

If you are using non-standard field-names in .bib database and bib-style, you have to declare them by `\_CreateField {fieldname}`.

You can declare `\_SortingOrder` in the manner documented by librarian package.

User or author of the bib-style can create the hidden field which has a precedence while sorting names. Example:

```
\CreateField {sortedby}
\SpecialSort {sortedby}
```

Suppose that the .bib file includes:

```
... 
author = "Jan Chadima",
sortedby = "Hzzadima Jan",
...
```

Now, this author is sorted between H and I, because the Ch digraph in this name has to be sorted by this rule.

If you need (for example) to place the auto-citations before other citations, then you can mark your entries in .bib file by `sortedby = "@"`, because this character is sorted before A.

2.32.4 The `usebib.opm` macro file loaded when `\usebib` is used

Loading the `librarian.tex` macro package. See `texdoc librarian` for more information about it.

We want to ignore `\errmessage` macro and we want not to create `\jobname.lbr` file.

The `usebib` command.

```
\def\usebib/#1 (#2) #3 {%
  \let\citeI=\relax
  \xdef\citelist{\trycs{ctlst:\bibp}{}\trycs{ctlst\&:\bibp}{}}%
  \global \ea\let \csname _ctlst:\bibp\endcsname =\write
  \ifx\citelist\empty\else % there was \nocite\[*\] used.
    \begin{flushleft}
      \texttt{No cited items. \noexpand\usebib ignored}%
    \end{flushleft}
  \else
    \begin{flushleft}
      \texttt{No cited items. \noexpand\usebib ignored}%
    \end{flushleft}
  \fi
\end{flushleft}
```
Corrections in librarian macros.

Main action per each entry.
Various macros + multilingual. Note that \nobibwarnlist is used in \bibwarning and it is set by \nobibwarning macro.
2.32.5 Usage of the *bib-iso690* style

This is the iso690 bibliographic style used by OpTEX.

See *op-biblist.bib* for an example of the *.bib* input. You can try it by:

```latex
\fontfam[LMfonts]
\nocite[*]
\usebib/s (iso690) op-biblist
```

### Common rules in *.bib* files

There are entries of type `@FOO{...}` in the *.bib* file. Each entry consists of fields in the form `name = "value"`, or `name = {value}`. No matter which form is used. If the value is pure numeric then you can say simply `name = value`. Warning: the comma after each field value is mandatory! If it is missing then the next field is ignored or badly interpreted.

The entry names and field names are case insensitive. If there exists a data field no mentioned here then it is simply ignored. You can use it to store more information (abstract, for example).

There are “standard fields” used in ancient bibTEX (author, title, editor, edition, etc., see [http://en.wikipedia.org/wiki/BibTeX](http://en.wikipedia.org/wiki/BibTeX)). The *iso690* style introduces several “non-standard” fields: ednote, numbering, isbn, issn, doi, url, citedate, key, bibmark. They are documented here.

Moreover, there are two optional special fields:

- `lang = language of the entry. The hyphenation plus autogenerated phrases and abbreviations will be typeset by this language.`
- `option = options by which you can control a special printing of various fields.`

There can be only one option field per each entry with (maybe) more options separated by spaces. You can declare the global option(s) in your document applied for each entry by \`\biboptions={...}`.

### The author field

All names in the author list have to be separated by “and”. Each author can be written in various formats (the *von* part is typically missing):

- Firstname(s) von Lastname
- von Lastname, Firstname(s)
- von Lastname, After, Firstname(s)

Only the Lastname part is mandatory. Examples:

- Petr Olšák
- Olšák, Petr
- Leonardo Piero da Vinci
- da Vinci, Leonardo Piero
- da Vinci, painter, Leonardo Piero

The separator “and” between authors will be converted to comma during printing, but between the semifinal and final author the word “and” (or something different depending on the current language) is printed.

The first author is printed in reverse order: “LASTNAME, Firstname(s) von, After” and the other authors are printed in normal order: “Firstname(s) von LASTNAME, After”. This feature follows the ISO 690 norm. The Lastname is capitalized using uppercase letters. But if the \`\caps` font modifier is defined, then it is used and printed `\caps\_rm␣Lastname`.

160
You can specify the option `aumax:⟨number⟩`. The ⟨number⟩ denotes the maximum authors to be printed. The rest of the authors are ignored and the et-al. is appended to the list of printed authors. This text is printed only if the `aumax` value is less than the real number of authors. If you have the same number of authors in the .bib file as you need to print but you want to append et-al, then you can use `auetal` option.

There is an `aumin:⟨number⟩` option which denotes the definitive number of printed authors if the author list is not fully printed due to `aumax`. If `aumin` is unused then `aumax` authors are printed in this case.

All authors are printed if `aumax:⟨number⟩` option isn’t given. There is no internal limit. But you can set the global options in your document by setting the \biboptions tokens list. For example:

\biboptions={aumax:7 aumin:1}
% if there are 8 or more authors then only the first author is printed.

Examples:

author = "John Green and Bob Brown and Alice Black",
output: GREEN, John, Bob BROWN, and Alice BLACK.

author = "John Green and Bob Brown and Alice Black",
option = "aumax:1",
output: GREEN, John et al.

author = "John Green and Bob Brown and Alice Black",
option = "aumax:2",
output: GREEN, John, Bob BROWN et al.

author = "John Green and Bob Brown and Alice Black",
option = "aumax:3",
output: GREEN, John, Bob BROWN, and Alice BLACK.

author = "John Green and Bob Brown and Alice Black",
option = "auetal",
output: GREEN, John, Bob BROWN, Alice BLACK et al.

If you need to add a text before or after the author’s list, you can use the `auprint:⟨value⟩` option. The ⟨value⟩ will be printed instead of the authors list. The ⟨value⟩ can include \AU macro which expands to the authors list. Example:

author = "Robert Calbraith",
option = "auprint:{\AU\space [pseudonym of J. K. Rowling]}",
output: CALBRAITH Robert [pseudonym of J. K. Rowling].

You can use the `autrim:⟨number⟩` option. All Firstnames of all authors are trimmed (i.e. reduced to initials) iff the number of authors in the author field is greater than or equal to ⟨number⟩. There is an exception: `autrim:0` means that no Firstnames are trimmed. This is the default behavior. Another example: `autrim:1` means that all Firstnames are trimmed.

author = "John Green and Bob Brown and Alice Black",
option = "auetal autrim:1",
output: GREEN, J., B. BROWN, A. BLACK et al.

If you need to write a team name or institution instead of authors, replace all spaces by \␣ in this name. Such text is interpreted as Lastname. You can add the secondary name (interpreted as Firstname) after the comma. Example:

author = "Czech\ Technical\ University\ in\ Prague, Faculty\ of\ Electrical\ Engineering",
output: CZECH TECHNICAL UNIVERSITY IN PRAGUE, Faculty of Electrical Engineering.

The editor field
The editor field is used for the list of the authors of the collection. The analogous rules as in author field are used here. It means that the authors are separated by “ and “, the Firstnames, Lastnames, etc. are
interpreted and you can use the options \texttt{edmax}:\langle\text{number}\rangle, \texttt{edmin}:\langle\text{number}\rangle, \texttt{edetal}, \texttt{edtrim}:\langle\text{number}\rangle and \texttt{edprint}:%\langle\text{value}\rangle\rangle (with \textsc{\texttt{ED}} macro). Example:

\begin{verbatim}
editor = "Jan Tomek and Petr Karas",
option = "edprint:\langle\text{\textsc{ED}}, editors.}\rangle edtrim:1",
\end{verbatim}

Output: J. TOMEK and P. KARAS, editors.

If \texttt{edprint} option is not set then \langle\text{\textsc{ED}}, \texttt{eds.}\rangle or \langle\text{\textsc{ED}}, \texttt{ed.}\rangle is used depending on the entry language and on the singular or plural of the editor(s).

\textbf{The ednote field}

The \texttt{ednote} field is used as the secondary authors and more editorial info. The value is read as raw data without any interpretation of Lastname, Firstname etc.

\begin{verbatim}
ednote = "Illustrations by Robert Agarwal, edited by Tom Nowak",
\end{verbatim}

output: Illustrations by Robert AGARWAL, edited by Tom NOWAK.

The \texttt{upper} command has to be used for Lastnames in the \texttt{ednote} field.

\textbf{The title field}

This is the title of the work. It will be printed (in common entry types) by italics. The ISO 690 norm declares, that the title plus optional subtitle are in italics and they are separated by a colon. Next, the optional secondary title has to be printed in an upright font. This can be added by \texttt{titlepost}:\langle\text{\langle\text{value}\rangle}\rangle. Example:

\begin{verbatim}
title = "The Simple Title of The Work",
or 
title = "Main Title: Subtitle",
or 
title = "Main Title: Subtitle",
option = "titlepost:Secondary title",
\end{verbatim}

The output of the last example: \textit{Main Title: Subtitle}. Secondary title.

\textbf{The edition field}

This field is used only for second or more edition of cited work. Write only the number without the word "edition". The shortcut "ed." (or something else depending on the current language) is added automatically. Examples:

\begin{verbatim}
edition = "Second",
edition = "2nd",
edition = "2\$\langle\text{\texttt{\textsc{ED}} nd}\rangle$",
edition = "2."
\end{verbatim}

Output of the last example: 2. ed.

\begin{verbatim}
edition = "2."
lang = "cs",
\end{verbatim}

Output: 2. vyd.

Note, that the example \texttt{edition=}\langle\text{\langle\text{value}\rangle}\rangle may cause problems. If you are using language "cs" then the output is bad: Second vyd. But you can use \texttt{editionprint}:\langle\text{\langle\text{value}\rangle}\rangle option. The the \langle\text{\langle\text{value}\rangle}\rangle is printed instead of edition field and shortcut. The edition field must be set. Example:

\begin{verbatim}
edition = "whatever",
option = "editionprint:Second full revised edition",
\end{verbatim}


You can use \texttt{\textsc{\texttt{EDN}}} macro in \texttt{editionprint} value. This macro is expanded to the edition value. Example:

\begin{verbatim}
edition = "Second",
option = "editionprint:\langle\text{\textsc{\texttt{EDN}}} space full revised edition}\rangle",
or 
edition = "Second full revised edition",
option = "editionprint:\langle\text{\textsc{\texttt{EDN}}}\rangle",
\end{verbatim}

162
The address, publisher, year fields
This is an anachronism from ancient BibTeX (unfortunately no exclusive) that the address field includes only the city of the publisher’s residence. No more data are here. The publisher field includes the name of the publisher.

    address = "Berlin",
publisher = "Springer Verlag",
year = 2012,


Note, that the year needn’t to be inserted into quotes because it is pure numeric.
The letter a, b, etc. are appended to the year automatically if two or more subsequent entries in the bibliography list are not distinct by the first author and year fields. If you needn’t this feature, you can use the noautoletters option.

You can use "yearprint:{value}" option. If it is set then the ⟨value⟩ is used for printing year instead the real field value. The reason: year is sort sensitive, maybe you need to print something else than only sorting key. Example:

    year = 2000,
    option = "yearprint:{© 2000}",


    year = "2012a",
    option = "yearprint:{2012}",


The address, publisher, and year are typically mandatory fields. If they are missing then the warning occurs. But you can set unpublished option. Then this warning is suppressed. There is no difference in the printed output.

The url field
Use it without \url macro, but with http:// prefix. Example:

    url = "http://petr.olsak.net/opmac.html",

The ISO 690 norm recommends to add the text “Available from” (or something else if a different current language is used) before URL. It means, that the output of the previous example is:

If the cs language is the current one than the output is:

    Dostupné z: http://petr.olsak.net/opmac.html.
If the urlalso option is used, then the added text has the form “Available also from” or “Dostupné také z:” (if cs language is current).

The citedate field
This is the citation date. The field must be in the form year/month/day. It means, that the two slashes must be written here. The output depends on the current language. Example:

    citedate = "2004/05/21",
Output when en is current: [cit. 2004-05-21].
Output when cs is current: [vid. 21. 5. 2004].

The howpublished field
This declares the available medium for the cited document if it is not in printed form. Alternatives: online, CD, DVD, etc. Example:

    howpublished = "online",
Output: [online].

The volume, number, pages and numbering fields
The volume is the “big mark” of the journal issue and the number is the “small mark” of the journal issue and pages includes the page range of the cited article in the journal. The volume is prefixed by Vol., the number by No., and the pages by pp. But these prefixes depends on the language of the entry.

    example:

163
volume = 31,
number = 3,
pages = "37--42",

volume = 31,
number = 3,
pages = "37--42",
lang = "cs",
Output: ročník 31, č. 3, s. 37–42.

If you disagree with the default prefixes, you can use the numbering field. When it is set then it is used instead of volume, number, pages fields and instead of any mentioned prefixes. The numbering can include macros \VOL, \NO, \PP, which are expanded to the respective values of fields. Example:

volume = 31,
number = 3,
pages = "37--42",
numbering = "Issue~\VOL/\NO, pages~\PP",
Output: Issue 31/3, pages 37–42

Note: The volume, numbers, and pages fields are printed without numbering filed only in the \ARTICLE entry. It means, that if you need to visible them in the \INBOOK, \INPROCEEDINGS etc. entries, then you must use the numbering field.

Common notes about entries
The order of the fields in the entry is irrelevant. We use the printed order in this manual. The exclamation mark (!) denotes the mandatory field. If the field is missing then a warning occurs during processing.

If the unpublished option is set then the fields address, publisher, year, isbn, and pages are not mandatory. If the nowarn option is set then no warnings about missing mandatory fields occur.

If the field is used but not mentioned in the entry documentation below then it is silently ignored.

• The \BOOK entry
This is used for book-like entries.
Fields: author(!), title(!), howpublished, edition, ednote, address(!), publisher(!), year(!), citedate, series, isbn(!), doi, url, note.
The ednote field here means the secondary authors (illustrator, cover design etc.).

• The \ARTICLE entry
This is used for articles published in a journal.
Fields: author(!), title(!), journal(!), howpublished, address, publisher, month, year, [numbering or volume, number, pages(!)], citedate, issn, doi, url, note.
If the numbering is used then it is used instead volume, number, pages.

• The \INBOOK entry
This is used for the part of a book.
Fields: author(!), title(!), booktitle(!), howpublished, edition, ednote, address(!), publisher(!), year(!), numbering, citedate, series, isbn or issn, doi, url, note.
The author field is used for author(s) of the part, the editor field includes author(s) or editor(s) of the whole document. The pages field specifies the page range of the part. The series field can include more information about the part (chapter numbers etc.).
The \INPROCEEDINGS and \CONFERENCE entries are equivalent to \INBOOK entry.

• The \THESIS entry
This is used for the student’s thesis.
Fields: author(!), title(!), howpublished, address(!), school(!), month, year(!), citedate, type(!), ednote, doi, url, note.
The type field must include the text “Master’s Thesis” or something similar (depending on the language of the outer document).
There are nearly equivalent entries: \BACHELORTHESIS, \MASTERSTHESIS and \PHDTHESIS. These entries set the type field to an appropriate value automatically. The type field is optional in this case. If it is used then it has precedence before the default setting.
• The @MISC entry
  It is intended for various usage.
  Fields: author, title, howpublished, ednote, citedate, doi, url, note.
  You can use \AU, \ED, \EDN, \VOL, \NO, \PP, \ADDR, \PUBL, \YEAR macros in ednote field. These macros print authors list, editors list, edition, volume, number, pages, address, publisher, and year field values respectively.
  The reason for this entry is to give to you the possibility to set the format of entry by your own decision. The most of data are concentrated in the ednote field.

• The @BOOKLET, @INCOLLECTION, @MANUAL, @PROCEEDINGS, @TECHREPORT, @UNPUBLISHED entries
  These entries are equivalent to @MICS entry because we need to save the simplicity. They are implemented only for (almost) backward compatibility with the ancient Bib\TeX. But the ednote is mandatory field here, so you cannot use these entries from the old databases without warnings and without some additional work with the .bib file.

The cite-marks (bibmark) used when \nonumcitations is set
When \nonumcitations is set then \cite prints text-oriented bib-marks instead of numbers. This style file auto-generates these marks in the form “Lastname of the first author, comma, space, the year” if the bibmark field isn’t declared. If you need to set an exception from this common format, then you can use bibmark field.

The OPmac trick http://petr.olsak.net/opmac-tricks-e.html#bibmark describes how to redefine the algorithm for bibmark auto-generating when you need the short form of the type [Au13].

Sorting
If \usebib/c is used then entries are sorted by citation order in the text. If \usebib/s is used then entries are sorted by “Lastname, Firstname(s)” of the first author and if more entries have this value equal, then the year is used (from older to newer). This feature follows the recommendation of the ISO 690 norm.

If you have the same authors and the same year, you can control the sorting by setting years like 2013, 2013a, 2013b, etc. You can print something different to the list using \yearprint{⟨value⟩} option, see the section about address, publisher, and year above. The real value of year field (i.e. not yearprint value) is also used in the text-oriented bib-marks when \nonumcitations is set.

If you have some problems with name sorting, you can use the hidden field key, which is used for sorting instead of the “Lastname Firstname(s)” of authors. If the key field is unset then the “Lastname Firstname(s)” is used for sorting normally. Example:

author = "Světla Čmejrková",
key = "Czzmejrkova Svetla",

This entry is now sorted between C and D.

The norm recommends placing the auto-citations at the top of the list of references. You can do this by setting key_{\text{\(\_\)A}} = "@", to each entry with your name because the @ character is sorted before A.

Languages
There is the language of the outer document and the languages of each entry. The ISO 690 norm recommends that the technical notes (the prefix before URL, the media type, the “and” conjunction between the semifinal and final author) maybe printed in the language of the outer document. The data of the entry have to be printed in the entry language (edition ed./vyd., Vol./ročník, No./č. etc.). Finally, there are the phrases independent of the language (for example In:). Unfortunately, the bib\TeX supposes that the entry data are not fully included in the fields so the automaton has to add some text during processing (“ed.”, “Vol.”, “see also”, etc.). But what language has to be chosen?

The current value of the \language register at the start of the .bib processing is described as the language of the outer document. This language is used for technical notes regardless of the entry language. Moreover, each entry can have the lang field (short name of the language). This language is used for ed./vyd., vol./ročník, etc. and it is used for hyphenation too. If the lang is not set then the outer document language is used.

You can use \Mtext{bib.⟨identifier⟩} if you want to use a phrase dependent on outer document language (no on entry language). Example:

howpublished = "\Mtext{bib.blue-ray}"

Now, you can set the variants of bib.blue-ray phrase for various languages:
Summary of non-standard fields
This style uses the following fields unknown by bib\TeX:  
  \begin{itemize}
    \item \texttt{option} \texttt{... options separated by spaces}
    \item \texttt{lang} \texttt{... the language two-letter code of one entry}
    \item \texttt{ednote} \texttt{... edition info (secondary authors etc.) or global data in @MISC-like entries}
    \item \texttt{citedate} \texttt{... the date of the citation in year/month/day format}
    \item \texttt{numbering} \texttt{... format for volume, number, pages}
    \item \texttt{isbn} \texttt{... ISBN}
    \item \texttt{issn} \texttt{... ISSN}
    \item \texttt{doi} \texttt{... DOI}
    \item \texttt{url} \texttt{... URL}
  \end{itemize}

Summary of options
  \begin{itemize}
    \item \texttt{aumax}:\langle \texttt{number} \rangle \texttt{... maximum number of printed authors}
    \item \texttt{aumin}:\langle \texttt{number} \rangle \texttt{... number of printed authors if aumax exceeds}
    \item \texttt{autrim}:\langle \texttt{number} \rangle \texttt{... full Firstnames iff number of authors are less than this}
    \item \texttt{auprint}:\{\langle \texttt{value} \rangle \} \texttt{... text instead authors list (\texttt{\AU} macro may be used)}
    \item \texttt{edmax}, \texttt{edmin}, \texttt{edtrim} \texttt{... similar as above for editors list}
    \item \texttt{edprint}:\{\langle \texttt{value} \rangle \} \texttt{... text instead editors list (\texttt{\ED} macro may be used)}
    \item \texttt{titlepost}:\{\langle \texttt{value} \rangle \} \texttt{... text after title}
    \item \texttt{yearprint}:\{\langle \texttt{value} \rangle \} \texttt{... text instead real year (\texttt{\YEAR} macro may be used)}
    \item \texttt{editionprint}:\{\langle \texttt{value} \rangle \} \texttt{... text instead of real edition (\texttt{\EDN} macro may be used)}
    \item \texttt{urlalso} \texttt{... the \texttt{``available also from''} is used instead \texttt{``available from''}}
    \item \texttt{unpublished} \texttt{... the publisher etc. fields are not mandatory}
    \item \texttt{nowarn} \texttt{... no mandatory fields}
  \end{itemize}

Other options in the option field are silently ignored.

2.32.6 Implementation of the bib-iso690 style

\begin{verbatim}
\_maybedot (alias \_ in the style file group) does not put the second dot.
\end{verbatim}
Formatting of Author/Editor lists.

\_def\_firstauthorformat{%  
\_upper\_{\_Lastname}\_bprint\_{\_Firstname}*, *\_bprint\_{\_Von}*, *%  
}  
\_def\_otherauthorformat{%  \_bprint\_{\_Firstname}* \_bprint\_{\_Von}* \_upper\_{\_Lastname}\_bprint\_{\_Junior}*, *%  
}  
\_def\_commonname{%  \_ifnum\_NameCount=1 \_firstauthorformat \_ifx\_dobibmark\_undefined \_edef\_dobibmark{\_Lastname}\_fi \_else \_ifnum\_namecount=\_NameCount \_ifx\_maybeetal\_empty \_bibconjunctionand\_else \_fi \_else \_fi \_otherauthorformat \_fi \_fi  
\_def\_authorname{%  \_ifnum\_NameCount>0\_namecount\_relax \_else \_commonname \_fi \_ifnum\_NameCount=0\_namecount\_relax \_maybeetal \_fi \_fi  
\_let\_editorname=\_authorname  
\_def\_prepareauedoptions#1{%  \_def\_mabyetal{}\csname lb@abbreviatefalse\_endcsname \_biboptionvalue{#1max}\_authormax \_biboptionvalue{#1min}\_authormin \_biboptionvalue{#1pre}\_authorpre \_biboptionvalue{#1print}\_authorprint \_isbiboption{#1etal}\_iftrue \_def\_maybeetal{\_Mtext{bib.etal}}\_fi \_biboptionvalue{#1trim}\_autrim \_let\_namecountraw=\_namecount \_ifx\_authormax\_empty \_else \_ifnum0\_authormax<0\_namecount \_edef\_namecount{\_ifx\_authormin\_empty\_authormax\_else\_authormin\_fi}\_%  \_def\_maybeetal{\_Mtext{bib.etal}}\%  \_fi\_fi \_ifx\_autrim\_empty \_def\_autrim{10000}\_fi \_ifnum\_autrim=0 \_def\_autrim{10000}\_fi \_ifnum0\_namecount<\_autrim\_relax \_else \_AbbreviateFirstname \_fi  
\_def\_maybeetal{}  
\_ifx\upper\_undefined \_ifx\caps\_undefined \_def\upper\{\_uppercase\_ea\}\_else \_def\upper#1{\_caps\_rm#1}\_fi \_fi  
\fi  
\let\upper=\upper  
Preparing bib-mark (used when \nonumcitations is set).  
\_def\_setbibmark{%  \_ifx\_dobibmark\_undefined \_def\_dobibmark{}\_fi \_RetrieveFieldIn{bibmark}\_tmp \_ifx\_tmp\_empty \_RetrieveFieldIn{year}\_tmp \_edef\_tmp{\_dobibmark, \_tmp}\_fi \_bibmark=\_ea\{\_tmp\}\_fi  
}  
Setting phrases.  
\_def\_bibconjunctionand{\_Mtext{bib.and}}  
\_def\_preurl{\_Mtext{bib.available}}  
\_let\_predoi=\_preurl  
\_def\_postedition{\_mtext{bib.edition}}  
\_def\_Inclause{In:~}  
\_def\_prevolume{\_mtext{bib.volume}}  
\_def\_prenumber{\_mtext{bib.number}}  
\_def\_prepages{\_mtext{bib.prepages}}  
\_def\_posteditor{\_ifnum0\_namecountraw>1 \_Mtext{bib.editors}\_else\_\_Mtext{bib.editor}\_\_fi}
\_Mtext\{(identifier)\} expands to a phrase by outer document language (no entry language).

Non-standard field names.

Sorting.

Supporting macros.

Entry types.
2.33 Sorting and making Index

\makeindex implements sorting algorithm at TeX macro-language level. You need not any external program.

There are two passes in the sorting algorithm. The primary pass does not distinguish between a group of letters (typically non-accented and accented). If the result of comparing two string is equal in primary pass then the secondary pass is started. It distinguishes between variously accented letters. Czech rules, for example, says: not accented before dieresis before acute before circumflex before ring. At less priority: lowercase letters must be before uppercase letters.

The \texttt{\_sortingdata\langle iso-code\rangle} implements these rules for the language \texttt{\langle iso-code\rangle}. The groups between commas are not distinguished in the first pass. The second pass distinguishes all characters mentioned in the \texttt{\_sortingdata\langle iso-code\rangle} (commas are ignored). The order of letters in the \texttt{\_sortingdata\langle iso-code\rangle} macro is significant for the sorting algorithm. The Czech rules (\texttt{cs}) are implemented here:

\begin{verbatim}
\def \_sortingdata\langle iso-code\rangle
/{, }-,.,@,\%
/aa\%a,%
/bB,%
/cC,%
/\%%
/dD\%
\end{verbatim}
Characters ignored by the sorting algorithm are declared in \_\_ignoredchars\{(iso-code)\}. The compound characters (two or more characters interpreted as one character in the sorting algorithm) are mapped to single invisible characters in \_\_compoundchars\{(iso-code)\}. Czech rules declare ch or Ch or CH as a single letter sorted between H and I. See \_\_sortingdata\{iso-code\} above where these declared characters are used.

The characters declared in \_\_ignoredchars\{} are ignored in the first pass without additional condition. All characters are taken into account in second pass: ASCII characters with code < 65 are sorted first if they are not mentioned in the \_\_sortingdata\{iso-code\} macro. Others not mentioned characters have undefined behavior during sorting.

\makeindex.opm

\def \_\_ignoredchars {,.;?!:'"|(()\[
\def \_\_compoundchars {ch:\^^T Ch:\^^U CH:\^^V} % DZ etc. are sorted normally

Slovak sorting rules are the same as Czech. The macro \_\_sortingdata\{iso-code\} includes Slovak letters too. Compound characters are the same. English sorting rules can be defined by \_\_sortingdata\{iso-code\} too because English alphabet is a subset of the Czech and Slovak alphabets. Only difference: \_\_compoundchars\{iso-code\} is empty in English rules.

You can declare these macros for more languages if you wish to use \makeindex with sorting rules with respect to your language. Note: if you need to map compound characters to a character, don’t use \"I or \"M because these characters have very specific category codes. And use space to separate more mappings, like in \_\_compoundchars\{iso-code\} above.

\makeindex.opm

Preparing to primary pass is implemented by the \_\_setprimarysorting macro. It is called from \makeindex macro and all processing of sorting is in a group.

\makeindex.opm
Preparing to secondary pass is implemented by the \_setsecondarysorting macro.

Strings to be sorted are prepared in ⟨string⟩ control sequences (to save \TeX memory). The \_preparesorting \langle string ⟩ converts ⟨string⟩ to \_tmpb with respect to the data initialized in \_setprimarysorting or \_setsecondarysorting. The compound characters are converted to single characters by the \_docompound macro.

Macro \_isAleB \langle string1 ⟩ \langle string2 ⟩ returns the result of comparison of given two strings to \_ifAleB control sequence. Usage: \_isAleB \langle string1 ⟩ \langle string2 ⟩ \_ifAleB ... \_else ... \_fi The converted strings (in respect of the data prepared for first pass) must be saved as values of \_string1 and \_string2 macros. The reason is speed: we don’t want to convert them repeatedly in each comparison. The macro \_testAleB \langle converted string1 ⟩ \_relax \langle converted string2 ⟩ \_relax \langle string1 ⟩ \langle string2 ⟩ does the real work. It reads the first character from both converted strings, compares them and if it is equal then calls itself recursively else gives the result.
Merge sort is very effectively implemented by \TeX{} macros. The following code is created by my son Miroslav. The \texttt{\mergesort} macro expects that all items in \texttt{\iilist} are separated by a comma when it starts. It ends with sorted items in \texttt{\iilist} without commas. So \texttt{\dosorting} macro must prepare commas between items.

\begin{verbatim}
\def\mergesort #1#2,#3{% by Miroslav Olsak
  \ifx,#1% % prazdna-skupina,neco, (#2=neco #3=pokracovani)
    \addto\iilist{#2,}% % dvojice skupin vyresena
    \sortreturn{\fif\mergesort#3}% % \mergesort pokracovani
  \fi
  \ifx,#3% % neco,prazna-skupina, (#1#2=neco #3=,)
    \addto\iilist{#1#2,}% % dvojice skupin vyresena
    \sortreturn{\fif\mergesort}% % \mergesort dalsi
  \fi
  \ifx\_end#3% % neco,konec (#1#2=neco)
    \ifx\_empty\iilist % neco=kompletni setrideny seznam
      \def\iilist{#1#2}%
      \sortreturn{\fif\fif % spojim \indexbuffer+necoa cele znova
        \edef\iilist{\ea,\ea\mergesort\iilist#1#2,#3}}%
      \fi\fi % zatriduji: p1+neco1,p2+neco2, (#1#2=p1+neco1 #3=p2)
    \_isAleB \_ifAleB % p1<p2
      \addto\iilist{#1}% % p1 do bufferu
      \sortreturn{\fif\mergesort#2,#3}% % \mergesort p1+neco1,neco2,
    \else % p1>p2
      \addto\iilist{#3}% % p2 do bufferu
      \sortreturn{\fif\mergesort#1#2,}% % \mergesort neco1,p2+neco2,
    \fi
    \relax % zarazka, na ktere se zastavi \sortreturn\fi\fi
\end{verbatim}

The \texttt{\dosorting} \texttt{\list} macro redefines \texttt{\list} as sorted \texttt{\list}. The \texttt{\list} have to include control sequences in the form \texttt{⟨c⟩⟨string⟩}. These control sequences will be sorted with respect to \texttt{⟨strings⟩} without change of meanings of these control sequences. Their meanings are irrelevant when sorting. The first character \texttt{⟨c⟩} in \texttt{⟨c⟩⟨string⟩} should be whatever. It does not influence the sorting. Op\TeX{} uses comma at this place for sorting indexes: \texttt{⟨word1⟩,⟨word2⟩,⟨word3⟩ ...}.

The current language (chosen for hyphenation patterns) is used for sorting data. If the macro \texttt{\_sortinglang} is defined as \texttt{⟨iso-code⟩} (for example \texttt{\def\_sortinglang{de}}) then this has precedence and current language is not used. Moreover, if you specify \texttt{\_asciisortingtrue} then ASCII sorting will be processed and all language sorting data will be ignored.

\begin{verbatim}
\newifi\_ifasciisorting\_asciisortingfalse
\def\dosorting #1{%
  \begingroup
    \_def\_nold\%\n    \_ifx\_sortinglang\_undefined \_edef\_sortinglang{\cs{lan:\_the\language}}\_fi
    \_ifasciisorting
      \_edef\_sortinglang{ASCII}\%
      \_def\_preparesorting\#1{\_edef\_tmpb{\cs{ignorefirst\csstring\#1}}}\%
      \_let\_setsecondarysorting=\relax\%
    \else \_setprimarysorting\fi
    \relax % zarazka, na ktere se zastavi \sortreturn\fi\fi
\end{verbatim}
The \makeindex prints the index. First, it sorts the \_iilist second, it prints the sorted \_iilist, each item is printed using \_printindexitem.

\makeindex\_par
\_ifx\_iilist\_empty \_opwarning{index data-buffer is empty. TeX me again}\
\_incr\_unresolvedrefs
\_else
\_dosorting \_iilist % sorting \_iilist
\_bgroup
\_rightskip=0pt plus1fil \_exhyphenpenalty=10000 \_leftskip=\_iindent
\_ea\_xargs \_ea\_printindexitem \_iilist ;\_par
\_egroup
\_fi
\_public \makeindex ;

\makeindex\_par
\_ifx\_iilist\_empty \_opwarning{index data-buffer is empty. TeX me again}\
\_incr\_unresolvedrefs
\_else
\_dosorting \_iilist % sorting \_iilist
\_bgroup
\_rightskip=0pt plus1fil \_exhyphenpenalty=10000 \_leftskip=\_iindent
\_ea\_xargs \_ea\_printindexitem \_iilist ;\_par
\_egroup
\_fi
\_public \makeindex ;

\_printii \_word& does more intelligent work because we are working with words in the form \langle main-word\rangle/\langle sub-word\rangle/\langle sub-sub-word\rangle. The \everyii tokens register is applied before \noindent. User can declare something special here.

The \newiiletter{\letter} macro is empty by default. It is invoked if first letter of index entries is changed. You can declare a design between index entries here. You can try, for example:

\def\newiiletter#1#2{}\bigskip \hbox{\setfontsize{at15pt}\bf\uppercase{#1}}} \medskip

\_printii\_word& gets \_word in the form \langle pg\rangle:{\langle type\rangle}, \langle pg\rangle:{\langle type\rangle},...\langle pg\rangle:{\langle type\rangle} and it converts them to \langle pg\rangle, \langle pg\rangle, \langle from\rangle--\langle to\rangle, \langle pg\rangle etc. The same pages must be printed only once and continuous consequences of pages must be compressed to the form \langle from\rangle--\langle to\rangle. Moreover, the consequence is continuous only if all pages have the same \langle type\rangle. Empty \langle type\rangle is most common, pages with b \langle type\rangle must
be printed as bold and with \textit{⟨type⟩} as italics. Moreover, the \textit{⟨pg⟩} mentioned here are \textit{⟨gpageno⟩}, but we have to print \textit{⟨pageno⟩}. The following macros solve these tasks.

\begin{verbatim}
\def\printiipages#1&\{
  \let\pgtype=\undefined
  \tmpnum=0
  \printpages #1,:,\par
\}
\def\printpages#1:#2,{
  \ifx#1,\uselastpgnum
  \else \edef\tmpa{#2}%
  \ifx\pgtype\tmpa \else
    \let\pgtype=\tmpa
    \uselastpgnum \usepgcomma \pgprint#1:{#2}%
  \fi
  \tmpnum=#1 \returnfi \fi
  \ifnum\tmpnum=#1 \returnfi \fi
  \advance\tmpnum by1
  \ifnum\tmpnum=\lastpgnum
    \usepgdash \fi
  \edef\lastpgnum{\the\tmpnum:{\pgtype}}%
  \returnfi \fi
  \usepgcomma \pgprint#1:{#2}%
  \tmpnum=#1 \relax
  \ea\printpages \fi
\def\returnfi #1\relax{\fi}
\def\uselastpgnum{
  \ifx\lastpgnum\undefined
    \else \ea\pgprint\lastpgnum \let\lastpgnum=\undefined \fi
  \}
\def\usepgcomma{
  \ifnum\tmpnum>0, \fi}
\def\usepgdash{\hbox{--}}
\def\pgprint #1:#2{%
  \ifx ,#2,\pgprintA{#1}\returnfi \fi
  \ifx b#2{\bf \pgprintA{#1}}\returnfi \fi
  \ifx i#2{\it \pgprintA{#1}}\returnfi \fi
  \ifx u#2\pgu{\pgprintA{#1}}\returnfi \fi
  \pgprintA{#1}\relax
\}
\def\pgprintA #1#2#3#4{% #1=<word> #2=<gpageno> #3=<pageno> #4=<iitype> \ifx#1\relax \global\addto\iilist {#1}% \gdef#1{#2:#4}% \else \global\addto#1{,#2:#4}% \fi
  \sxdef{\pgi:#2}{#3}%
\}
\def\xdef{\pgi:#2}{#3}%
\def\Xindex #1#2#3#4{% #1=<word> #2=<gpageno> #3=<pageno> #4=<iitype> \ifx#1\relax \global\addto\iilist {#1}% \gdef#1{#2:#4}% \else \global\addto#1{,#2:#4}% \fi
  \sxdef{\pgi:#2}{#3}%
\}
\def\xdef{\pgi:#2}{#3}%
\end{verbatim}

You can re-define \texttt{\_pgprint ⟨gpageno⟩}:{⟨iitype⟩} if you need to implement more \texttt{⟨iitypes⟩}.

\begin{verbatim}
\def\_pgprint #1:#2{%
  \ifx ,#2,\pgprintA{#1}\returnfi \fi
  \ifx b#2{\bf \pgprintA{#1}}\returnfi \fi
  \ifx i#2{\it \pgprintA{#1}}\returnfi \fi
  \ifx u#2\pgu{\pgprintA{#1}}\returnfi \fi
  \pgprintA{#1}\relax
\}
\def\pgprintA #1#2#3#4{% #1=<word> #2=<gpageno> #3=<pageno> #4=<iitype> \ifx#1\relax \global\addto\iilist {#1}% \gdef#1{#2:#4}% \else \global\addto#1{,#2:#4}% \fi
  \sxdef{\pgi:#2}{#3}%
\}
\def\xdef{\pgi:#2}{#3}%
\end{verbatim}

The implementation of macros \texttt{\_iindex}, \texttt{\_iindexd}, \texttt{\_iis} follows. Note that \texttt{\_iindex} works in the horizontal mode in order to the \texttt{\write} whatsit is not broken from the following word. If you need to keep vertical mode, use \texttt{\_iindex⟨word⟩} directly.

The \texttt{\_iindex⟨word⟩}:{⟨iitype⟩} stores \texttt{⟨word⟩} to the \texttt{\_iilist} if there is the first occurrence of the \texttt{⟨word⟩}. The list of pages where \texttt{⟨word⟩} occurs, is the value of the macro \texttt{\_iilist}, so \texttt{⟨gpageno⟩}:{⟨iitype⟩} is appended to this list. Moreover, we need a mapping from \texttt{⟨gpageno⟩} to \texttt{⟨pageno⟩}, because we print \texttt{⟨pageno⟩} in the index, but hyperlinks are implemented by \texttt{⟨gpageno⟩}. So, the macro \texttt{\_pgi} is defined as \texttt{⟨gpageno⟩}.

\begin{verbatim}
\def\_iilist {}\end{verbatim}

The implementation of macros \texttt{\_ii}, \texttt{\_iid}, \texttt{\_iis} follows. Note that \texttt{\_ii} works in the horizontal mode in order to the \texttt{\write} whatsit is not broken from the following word. If you need to keep vertical mode, use \texttt{\_iindex⟨word⟩} directly.

The \texttt{⟨iitype⟩} saves the \texttt{⟨type⟩} to the \texttt{\_iitypesaved} macro. It is used in the \texttt{\_iindex} macro.
2.34 Footnotes and marginal notes

\_gfnotenum is a counter which counts footnotes globally in the whole document. \_lfnotenum is a counter which counts footnotes at each chapter from one. It is used for local page footnote counters too. \_ifpgfnotenum says that footnote numbers are counted on each page from one. We need to run \openref in this case. \_fnotenum is a macro that expands to footnote number counted in declared part. \_fnotenumchapters declares footnotes numbered in each chapter from one (default), \_fnotenumglobal declares footnotes numbered in whole document from one and \_fnotenumpages declares footnotes numbered at each page from one.

The \_printfnotemark prints the footnote mark. You can re-define this macro if you want another design of footnotes. For example

\fnotenumpages
\def \_printfnotemark {$^\text{\_fnotenum}$} % default footnote mark
\def \_printfnotemarkA {$^\text{\_fnotenum}$} % footnote marks used in text
\def \_printfnotemarkB {$^\text{\_fnotenum}$} % footnote marks used in front of footnotes

This code gives footnotes* and ** and*** and† etc. and it supposes that there are no more than 6 footnotes at one page.

If you want to distinguish between footnote marks in the text and in the front of the footnote itself, then you can define \printfnotemarkA and \printfnotemarkB.

The \_fnotelinks⟨\textcolor{\_colorA}⟩⟨\textcolor{\_colorB}⟩ implements the hyperlinked footnotes (from text to footnote and backward).
Each footnote saves the \_Xfnote (without parameter) to the .ref file (if \openref). We can create the mapping from ⟨gfnotenum⟩ to ⟨pgfnotenum⟩ in the macro \_fn:⟨fnote⟩. Each \_Xpage macro sets the \_lfnotenum to zero.

By default \mnote{⟨text⟩} are in right margin at odd pages and they are in left margin at even pages. The \mnote macrosave its position to .ref file as \_Xmnote without parameter. We define \_mn:⟨mnotenum⟩ as \_right or \_left when the .ref file is read. The \ifnum 0 \leq 0#2 trick returns true if ⟨pageno⟩ has a numeric type and false if it is a non-numeric type (Roman numeral, for example). We prefer to use ⟨pageno⟩, but only if it has the numeric type. We use ⟨gpageno⟩ in other cases.

User can declare \fixmnotes\left or \fixmnotes\right. It defines \_mnotesfixed as \_left or \_right which declares the placement of all marginal notes and such declaration has a precedence.

The \_mnoteD{⟨text⟩} macro sets the position of the marginal note. The outer box of marginal note has zero width and zero depth and it is appended after current line using \vadjust primitive or it is inverted to vertical mode as a box shifted down by \parskip and with \vskip-\baselineskip followed.
The \mnoteskip is a dimen value that denotes the vertical shift of marginal note from its normal position. A positive value means shift up, negative down. The \mnoteskip register is set to zero after the marginal note is printed. The new syntax \mnote up\{\text{dimen}\}\{\text{text}\} is possible too, but public \mnoteskip is kept for backward compatibility.

The \mnoteA macro does the real work. The \lrmnote\{\langle\text{left}\rangle\}\{\langle\text{right}\rangle\} uses only first or only second parameter depending on the left or right marginal note.

We don’t want to process \fnote, \fnotemark, \mnote in TOC, headlines nor outlines.

2.35 Styles

OpTeX provides three styles: \report, \letter and \slides. Their behavior is documented in user part of the manual in the section 1.7.2 and \slides style (for presentations) is documented in \texttt{op-slides.pdf} which is an example of the presentation.

2.35.1 \report and \letter styles

We define auxiliary macro first (used by the \address macro)

The \{\text{boxlines}\}\{\langle line-1\rangle\\langle eol\rangle\langle line-2\rangle\\langle eol\rangle\ldots\langle line-n\rangle\\langle eol\rangle\} returns to the outer vertical mode a box with \langle line-1\rangle, next box with \langle line-2\rangle etc. Each box has its natural width. This is reason why we cannot use paragraph mode where each resulting box has the width \hsize. The \langle eol\rangle is set active and everypar starts \hbox{ and acive } \langle eol\rangle closes this \hbox by }.

The \report style initialization macro is defined here.
The \letter style initialization macro is defined here. The \letter defines \address and \subject macros. See the files demo/op-letter-*.tex for usage examples.

The \slides macro reads macro file slides.opm, see the section \ref{2.35.2}. Default margins and design is declared here. The \ttfont is scaled by mag1.15 in order to balance the ex height of Helvetica (Heros) and LM fonts Typewriter. The \begtt...\endtt verbatim is printed by smaller text.

2.35.2 \slides style for presentations

The bottom margin is set to 3mm. If we use 1mm, then the baseline of \footline is 2mm from the bottom page. This is the depth of the \Grey rectangle used for page numbers. It is r-lapped to \hoffset.
width because left margin = `\hoffset` = right margin. It is 14 mm for narrow pages or 16 mm for wide pages.

The `\subtit` is defined analogically like `\tit`.

The `\psshow(num)` prints the text in invisible (transparent) font when `\layernum<num>`. For transparency we need to define special graphics states.

The main level list of items is activated here. The `\item:X` and `\item:x` are used and are re-defined here. If we are in a nested level of items and `\pg*` is used then `\egroups` macro expands to the right number of `\egroup`s to close the page correctly. The level of nested item lists is saved to the `\ilevel` register and used when we start again the next text after `\pg*`.

The default values of `\pg`, i.e. `\pg;`, `\pg*` and `\pg.` are very simple. They are used when `\showslides` is not specified.

The `\endslides` is defined as `\end` primitive (preceeded by `\byehook`), but slide-designer can redefine it. For example, OpTeX trick 0029 shows how to define clickable navigation to the pages and how to check the data integrity at the end of the document using `\endslides`.

The `\bye` macro is redefined here as an alternative to `\pg*`.

We need no numbers and no table of contents when using slides. The `\printsec` macro is redefined in order the title is centered and typeset in `\Blue`.

180
When \texttt{\textbackslash slideshow} is active then each page is opened by \texttt{\setbox\textbackslash slidepage=\vbox\textbackslash bgroup} (roughly speaking) and closed by \texttt{\textbackslash egroup}. The material is \texttt{\unvbox}ed and saved for the usage in the next usage if \texttt{\pg+} is in process. The \texttt{\_slidelayer} is incremented instead of \texttt{\_pageno} if \texttt{\pg+}. This counter is equal to \texttt{\count1}, so it is printed to the terminal and log file next to \texttt{\_pageno}.

The code is somewhat more complicated when \texttt{\_layers} is used. Then \texttt{\textbackslash layered-text} is saved to the \texttt{\_layertext} macro, the material before it is in \texttt{\_slidepage} box and the material after it is in \texttt{\_slidepageB} box. The pages are completed in the \texttt{\loop} which increments the \texttt{\_layernum} register and prints page by page by the \texttt{\_printlayers} macro.
\_slidetext should be used instead \slideshow to deactivate it but keep the borders of groups.

When \slideshow is active then the destinations of internal hyperlinks cannot be duplicated to more “virtual” pages because hyperlink destinations have to be unique in the whole document.

The \slideshow creates boxes of typesetting material and copies them to more pages. So, we have to suppress creating destinations in these boxes. This is done in the \_slidelinks macro. We can move creating these destinations to the output routine. \_sdestbox is saved value of the original \_destbox which is redefined to do only \addto\_destboxes\_sdestbox\{\_label\}. All destinations saved to \_destboxes are created at the start of the next output routine in the \_pagedest macro. The output routine removes \_destboxes, so each destination is created only once.

Limitations of this solution: destinations are only at the start of the page, no at the real place where \wlabel was used. The first “virtual” page where \wlabel is used includes its destination. If you want to go to the final page of the partially uncovering ideas then use \wlabel\{\_label\}wlabel\_text in the last part of the page (before \pg;\) or use \pgref instead \ref.

The \_settinglayer is used in the \_layertext macro to prevent printing “Duplicate label” warning when it is expanded. It is done by special value of \_slideshook (used by the \wlabel macro). Moreover, the warning about illegal use of \bib, \usebib in \layers environment is activated.

Default \layers⟨num⟩ macro (when \slideshow is not activated) is simple. It prints the ⟨layered-text⟩ with \layernum=⟨num⟩+1 because we need the result after last layer is processed.

We must to redefine \notenumpages because the data from .ref file are less usable for implementing such a feature: the footnote should be in more layers repeatedly. But we can suppose that each page starts by \pg; macro, so we can reset the footnote counter by this macro.
2.36 Logos

Despite plain \TeX\ each macro for logos ends by \texttt{\_ignoreslash}. This macro ignores the next slash if it is present. You can use $\&\TeX/\$ like this for protecting the space following the logo. This is visually more comfortable. The macros $\&\TeX$, $\&\OpTeX$, $\&\LuaTeX$, $\&\XeTeX$ are defined.

The $\&\_slantcorr$ macro expands to the slant-correction of the current font. It is used to shifting A if the $\&\LaTeX$ logo is in italic.

The expandable versions of logos used in Outlines need the expandable \texttt{\_ignslash} (instead of the \texttt{\_ignoreslash}).

2.37 Multilingual support

2.37.1 Lowercase, uppercase codes

All codes in Unicode table keep information about pairs lowercase-uppercase letters or single letter. We need to read such information and set appropriate $\&\lccode$ and $\&\uccode$. The $\&\catcode$ above the code 127 is not set, i.e. the $\&\catcode=12$ for all codes above 127.

The file \texttt{UnicodeData.txt} is read if this file exists in your \TeX\ distribution. The format is specified at \url{http://www.unicode.org/L2/L1999/UnicodeData.html}. We read only Ll (lowercase letters), Lu (upperase letters) and Lo (other letters) and set appropriate codes. The scanner of \texttt{UnicodeData.txt} is
implemented here in the group (lines 6 to 15). After the group is closed then the file uni-lcuc.opm is left by \endinput.

If the file UnicodeData.txt does not exist then internal data are used. They follow to the end of the file uni-lcuc.opm.

\begin{verbatim}
\texttt{\input UnicodeData.txt} % If UnicodeData.txt not found, we have internal copy here from csplain, 2014:
\end{verbatim}

\section*{2.37.2 Hyphenations}

The \texttt{\textcircled{iso-code}} means a shortcut of language name (mostly by ISO 639-1). The following control sequences are used for language switching:

- \_lan:\langle number\rangle expands to \texttt{\textcircled{iso-code}} of the language. The \langle number\rangle is an internal number of languages used as a value of \texttt{\language} register.
- \_ulan:\langle long-lang\rangle expands to \texttt{\textcircled{iso-code}} too. This is transformation from long name of language (lowercase letters) to \texttt{\textcircled{iso-code}}.
- \_\langle iso-code\rangle Patt (for example \texttt{csPatt}) is the language \langle number\rangle declared by \texttt{\chardef}.
- \_\langle iso-code\rangle lang (for example \texttt{enlang, cslang, sklang, delang, pllang}) is language selector. It exists in two states
  - Initialization state: when \_\langle iso-code\rangle lang is used first then it must load the patterns into memory using Lua code. If it is done then the \_\langle iso-code\rangle lang re-defines itself to the processing state.
  - Processing state: it only sets \_\texttt{language=\langle iso-code\rangle Patt}, i.e. it selects the hyphenation patterns. It does a little more language-dependent work, as mentioned below.
- \_langspecific:\langle isocode\rangle is processed by \_\langle iso-code\rangle lang and it should include language-specific macros declared by the user or macro designer.
The USenglish patterns are preloaded first:

\def\lan{\enPatt=0}
\def\langlist{\en}(en)
\def\enlang{\enPatt=0}
\def\enlanglist{\en=lang}
\sdef{\lan:0}{en}
\sdef{\ulan:usenglish}{en}
\def\enlang{\lan}(en)
\input hyphen % en(USenglish) patterns from TeX82

\_preplang ⟨iso-code⟩ ⟨long-lang⟩ ⟨hyph-file-spec⟩ ⟨number⟩ ⟨pre-hyph⟩ ⟨post-hyph⟩ prepares the \⟨iso-code⟩ lang to its initialization state. Roughly speaking, it does:

\chardef\_⟨iso-code⟩ Patt = ⟨number⟩
\def\_lan:⟨number⟩ {⟨iso-code⟩}
\def\_ulan:⟨long-lang⟩ {⟨iso-code⟩}
\def\⟨iso-code⟩ lang {\loadpattrs ⟨hyph-file-spec⟩ ⟨number⟩ ⟨long-lang⟩ % loads patterns using Lua code
\edef\⟨iso-code⟩ lang {\uselang{⟨iso-code⟩}\langleiso-code⟩Patt ⟨pre-hyph⟩ ⟨post-hyph⟩} % runs itself in processing state}
\addto\langlist{ ⟨#1(#2)}%
\sdef{#1lang_ea}\ea{\csname _#1lang_endcsname} % unprefixed \langleisocode⟩lang

\loadpattrs ⟨hyph-file-spec⟩ ⟨number⟩ ⟨long-lang⟩ loads hyphenation patterns and hyphenation exceptions for given language and registers them as \language=⟨number⟩.

The ⟨hyph-file-spec⟩ is a part of full file name which is read: hyph-⟨hyph-file-spec⟩.tex. The patterns and hyphenation exceptions are saved here in UTF-8 encoding. The ⟨hyph-file-spec⟩ should be a list of individual ⟨hyph-file-spec⟩’s separated by commas, see the language Serbian below for an example.

\uselang{⟨iso-code⟩}\langleiso-code⟩Patt ⟨pre-hyph⟩ ⟨post-hyph⟩ sets \language, \lefthyphenmin, \righthyphenmin and runs \frenchspacing. This default language-dependent settings should be re-declared by \langspecific:⟨iso-code⟩ which is run finally (it is \relax by default, only \langspecific:en runs \nonfrenchspacing).
The \uselanguage \{⟨long-lang⟩\} is defined here (for compatibility with e-plain users).

The numbers for languages are declared as fixed constants (no auto-generated). This concept is inspired by CSplain. There are typical numbers of languages in CSplain: 5=Czech in IL2, 15=Czech in T1 and 115=Czech in Unicode. We keep these constants but we load only Unicode patterns (greater than 100), of course.
The \langlist includes names of all languages which are ready to load and use their hyphenation patterns. This list is printed to the terminal and to log at init\TeX state here. It can be used when processing documents too.

Maybe, you need to do more language-specific actions than just switching hyphenation patterns. For example, you need to load a specific font with a specific script used in the selected language, you can define macros for quotation marks depending on the language, etc.

The example shows how to declare such language-specific things.

```latex
\def\langset #1 #2{
  \sdef{\langspecific:#1}{#2}
}\langset fr {... declare French quotation marks}
\langset de {... declare German quotation marks}
\langset gr {... switch to Greek fonts family}
... etc.
```

Note that you need not set language-specific phrases (like \today) by this code. Another concept is used for such tasks. See the section 2.37.3 for more details.

### 2.37.3 Multilingual phrases and quotation marks

Only four words are generated by \OpTeX macros: “Chapter”, “Table”, “Figure” and “Subject”. These phrases can be generated depending on the current value of \language register, if you use \mtext⟨phrase-id⟩, specially \mtext[chap], \mtext[t], \mtext[f] or \mtext[subj]. If your macros generate more words then you can define such words by \sdef⟨phrase-id⟩:{lang} where ⟨phrase-id⟩ is a label for the declared word and ⟨lang⟩ is a language shortcut (iso code).
Using `\_langw` \{lang\} \{chapter\} \{table\} \{figure\} \{subject\} you can declare these words more effectively:

```latex
\_def \_langw #1 #2 #3 #4 #5 {%
  \_sdef{_mt:chap:#1}{#2}\_sdef{_mt:t:#1}{#3}\_sdef{_mt:f:#1}{#4}%
  \_sdef{_mt:subj:#1}{#5}%
}\_langw en Chapter Table Figure Subject
%--------------------------------------------------------------
\_langw cs Kapitola Tabulka Obrázek Věc
\_langw de Kapitel Tabelle Abbildung Betreff
\_langw es Capítulo Tabla Figura Sujeto
\_langw fr Chapitre Tableau Figure Matière
\_langw it Capitolo Tabella Fig. Oggetto
\_langw pl Rozdział Tabela Ilustracja Temat
... etc. (see languages.opm)
```
You can add more words as you wish. For example `\today` macro:

```latex
\_def \_monthw #1 #2 #3 #4 #5 #6 #7 {%
  \_sdef{_mt:m1:#1}{#2}\_sdef{_mt:m2:#1}{#3}\_sdef{_mt:m3:#1}{#4}%
  \_sdef{_mt:m4:#1}{#5}\_sdef{_mt:m5:#1}{#6}\_sdef{_mt:m6:#1}{#7}%
}\_monthwB #1\_monthwB #1 #2 #3 #4 #5 #6 #7 {%
  \_sdef{_mt:m7:#1}{#2}\_sdef{_mt:m8:#1}{#3}\_sdef{_mt:m9:#1}{#4}%
  \_sdef{_mt:m10:#1}{#5}\_sdef{_mt:m11:#1}{#6}\_sdef{_mt:m12:#1}{#7}%
}\_slet{_mt:today:sk}{_mt:today:cs}
\_def \_today\{\_mtext{today}\}
\_public \today ;
```
Quotes should be tagged by `\"` ⟨text⟩ `\"` and `\'` ⟨text⟩ `\'` if ⟨iso-code⟩ quotes is declared at beginning of the document (for example `\enquotes`). If not, then the control sequences `\"` and `\'` are undefined. Remember, that they are used in another meaning when the `\oldaccents` command is used. The macros `\"` and `\'` are not defined as `\protected` because we need their expansion when `\outlines` are created. User can declare quotes by `\quoteschars ⟨lqq⟩⟨rqq⟩⟨lq⟩⟨rq⟩`, where `⟨clqq⟩`...`⟨crqq⟩` are normal quotes and `⟨clq⟩`...`⟨crq⟩` are alternative quotes. or use `\altquotes` to swap between the meaning of these two types of quotes.

`\enquotes`, `\csquotes`, `\dequotes`, `\frquotes` etc. are defined here.

```latex
\_def \_enquotes {\_quoteschars "\"\"
\_def \_csquotes {\_quoteschars "\"\"
\_def \_frquotes {\_quoteschars "\"«»'
\_let \_plquotes = \_frquotes
\_let \_esquotes = \_frquotes
\_let \_grquotes = \_frquotes
\_let \_ruquotes = \_frquotes
\_let \_itquotes = \_frquotes
\_let \_skquotes = \_csquotes
\_let \_dequotes = \_csquotes
\_slet{_mt:today:en}{\_the\_day, \_the\_year}
\_sdef{mt:today:en}{\_the\_month\_today:en}
\_slet{_mt:today:cs}{\_the\_day, \_the\_month\_today:cs}
\_slet{_mt:today:sk}{\_the\_day, \_the\_month\_today:sk}
\_def \_today\{\_mtext{today}\}
\_public \today ;
```
```
The `\quoteschars ⟨lqq⟩⟨rqq⟩⟨lq⟩⟨rq⟩` defines `\"` and `\'` as `\_qqA` in in normal mode and as expadable macros in outline mode. We want to well process the common cases: `\"` ⟨``⟩ or `\"` ⟨``⟩. This is the
reason why the quotes parameter is read in verbatim mode and retokenized again by \textbackslash{scantextokens}. We want to allow to quote the quotes mark itself by \"{`}\". This is the reason why the sub-verbatim mode is used when the first character is \{ in the parameter.

The \" is defined as \_qqA\_qqB{\langle\textbackslash{ly}}\rangle and \' as \_qqA\_qqC{\langle\textbackslash{r text}\rangle}. The \_qqA\_qqB{\langle\textbackslash{text}\rangle} runs \_qqB{\langle\textbackslash{text}\rangle}\rangle. The \_qqA\_qqB runs \_qqB{\langle\textbackslash{text}\rangle}\rangle\". The \_regquotes" does \def\"#1\"{\langle\textbackslash{L}\rangle #1 \langle\textbackslash{R}\rangle} for outlines but the " separator is active (because " and ' are active in \textbackslash{pdfuniddef}).

Sometimes should be usable to leave the markup \"such\" or 'such' i.e. without the first backslash. Then you can make the characters " and ' active by the \textbackslash{activequotes} macro and leave quotes without the first backslash. First, declare \langle iso-code \rangle\quoteschars, then \altquotes (if needed) and finally \activequotes.

Bibliography references generated by \textbackslash{usebib} uses more language-dependent phrases. They are declared here. We don't want to save all these phrases into the format, so the trick with \textbackslash{endinput} is used here. When \textbackslash{usebib} is processed then the following part of the file \textbackslash{languages.opm} is read again.

Only phrases of few languages are declared here now. If you want to declare phrases of your language, please create an “issue” or a “request” at https://github.com/olsak/OpTeX or send me an email with new phrases for your language (or language you know). I am ready to put them here. Temporarily, you can put your definitions into \textbackslash{bibtexhook} token list.
2.38 Other macros

Miscellaneous macros are here.

\useOpTeX and \useoptex are declared as \relax.

\let \useOpTeX = \relax \let \useoptex = \relax

The \lastpage and \totalpages get the information from the \currpage. The \Xpage from .ref file sets the \currpage.

\def \totalpages {\openref\na\uses{second}\currpage}
\def \lastpage {\openref\na\ignoresecond\currpage}
\def \currpage {{0}{?}}
\public \lastpage \totalpages ;

We need \uv, \clqq, \crqq, \flqq, \frqq, \uslang, \ehyph, \chyph, \shyph, for backward compatibility with \csplain. Codes are set according to Unicode because we are using Czech only in Unicode when \LuaTeX is used.

% for compatibility with \csplain:
\chardef \clqq = 8222 \chardef \crqq = 8220
\chardef \flqq = 171 \chardef \frqq = 187
\chardef \promile = 8240
\let \uv = \clqq #1 \crqq
\let \uslang = \enlang \let \ehyph = \enlang
\let \chyph = \cslang \let \shyph = \sklang
\let \csUnicode = \csPatt \let \czUnicode = \csPatt \let \skUnicode = \skPatt

The \letfont was used in \csplain instead of \fontlet.

\letfont

Non-breaking space in Unicode.

\let \^a = ~

TikZ needs these funny control sequences.

\ea \toksdef \csname toks@endcsname\endcsname=0
\ea \let \csname voidb@\endcsname=\\voidbox

We don’t want to read opmac.tex unless \input opmac is specified.

\def\OPmacversion{OpTeX}

We allow empty lines in math formulae. It is more comfortable.

\suppressmathparerror = 1

Lorem ipsum can be printed by \lipsum[⟨range⟩] or \lorem[⟨range⟩], for example \lipsum[3] or \lipsum[112-121], max=150.

First usage of \lipsum reads the \LaTeX file lipsum.ltd.tex by \lipsumload and prints the selected paragraph(s). Next usages of \lipsum prints the selected paragraph(s) from memory. This second and more usages of \lipsum are fully expandable. If you want to have all printings of \lipsum expandable, use dummy \lipsum[0] first.

\lipsum adds \par after each printed paragraph. If you don’t need such \par here, use \lipsumtext[⟨number⟩]. This macro prints only one selected paragraph ⟨number⟩ and does not add \par.
We set \partokenname to \par in order to keep the name \par in user name space. I.e. a user can say \def \par {paragraph} for example without crash of processing the document. See section 2.2 for more details about the name space concept.

Moreover, we set \partokencontext to one in order to the \par token is inserted not only at empty lines, but also at the end of \vbox, \vtop and \vcenter if horizontal mode is opened here. This differs from default \TeX behavior where horizontal mode is closed in these cases without inserting par token.
We set \_partokenset to defined value 1 in order to the macro programmer can easily check these settings in OpTEX format by \ifx\_partokenset\undefined ... \else ...\fi.

```latex
\_partokenname\_par
\_partokencontext=1
\_let\_partokenset=1
\fi
```

2.39 Lua code embedded to the format

The file optex.lua is loaded into the format in optex.ini as byte-code and initialized by \everyjob, see section 2.1.

The file implements part of the functionality from luatexbase namespace, nowadays defined by LATEX kernel. luatexbase deals with modules, allocators, and callback management. Callback management is a nice extension and is actually used in OpTEX. Other functions are defined more or less just to suit luaotfload's use.

The allocations are declared in subsection 2.39.2, callbacks are implemented in subsection 2.39.3 and handling with colors can be found in the subsection 2.39.4.

2.39.1 General

Define namespace where some OpTEX functions will be added.

```lua
optex = optex or {}
```

Error function used by following functions for critical errors.

```lua
local function err(message)
  error("\error: \"..message..\n")
end
```

For a \chardef'd, \countdef'd, etc., csname return corresponding register number. The responsibility of providing a \XXdef'd name is on the caller.

```lua
function registernumber(name)
  return token.create(name).index
end
```

MD5 hash of given file.

```lua
function mdfive(file)
  local fh = io.open(file, "rb")
  if fh then
    local data = fh:read("*a")
    fh:close()
    tex.print(md5.sumhexa(data))
  end
end
```

2.39.2 Allocators

An attribute allocator in Lua that cooperates with normal OpTEX allocator.

```lua
alloc = alloc or {}
```

```lua
local attributes = {}
function alloc.new_attribute(name)
  local cnt = tex.count["_attributealloc"] + 1
  if cnt > 65534 then
    tex.error("No room for a new attribute")
  else
    tex.setcount("global", ".\_attributealloc", cnt)
    texio.write_nl("log", ".\..name.."\"\attribute\"..tostring(cnt))
    attributes[name] = cnt
    return cnt
  end
end
```
Allocator for Lua functions ("pseudoprimitives"). It passes variadic arguments ("...") like "global" to `token.set_lua`.

```lua
local function_table = lua.get_functions_table()
local luafnalloc = 0
function define_lua_command(csname, fn, ...)
    luafnalloc = luafnalloc + 1
    token.set_lua(csname, luafnalloc, ...) -- WARNING: needs LuaTeX 1.08 (2019) or newer
    function_table[luafnalloc] = fn
end
```

`provides_module` is needed by older version of luaotfload

```lua
provides_module = function() end
```

### 2.39.3 Callbacks

Save `callback.register` function for internal use.

```lua
local callback_register = callback.register
function callback.register(name, fn)
    err("direct registering of callbacks is forbidden, use 'callback.add_to_callback'")
end
```

Table with lists of functions for different callbacks.

```lua
local callback_functions = {}
```

Table that maps callback name to a list of descriptions of its added functions. The order corresponds with `callback_functions`.

```lua
local callback_description = {}
```

Table used to differentiate user callbacks from standard callbacks. Contains user callbacks as keys.

```lua
local user_callbacks = {}
```

Table containing default functions for callbacks, which are called if either a user created callback is defined, but doesn’t have added functions or for standard callbacks that are “extended” (see `mlist_to_hlist` and its pre/post filters below).

```lua
local default_functions = {}
```

Table that maps standard (and later user) callback names to their types.

```lua
local callback_types = {
    -- file discovery
    find_read_file = "exclusive",
    find_write_file = "exclusive",
    find_font_file = "data",
    find_output_file = "data",
    find_format_file = "data",
    find_vf_file = "data",
    find_map_file = "data",
    find_enc_file = "data",
    find_pk_file = "data",
    find_data_file = "data",
    find_opentype_file = "data",
    find_truetype_file = "data",
    find_type1_file = "data",
    find_image_file = "data",
    open_read_file = "exclusive",
    read_font_file = "exclusive",
    read_vf_file = "exclusive",
    read_map_file = "exclusive",
    read_enc_file = "exclusive",
    read_pk_file = "exclusive",
    read_data_file = "exclusive",
    read_truetype_file = "exclusive",
}
```
read_type1_file = "exclusive",
read_opentype_file = "exclusive",

-- data processing
process_input_buffer = "data",
process_output_buffer = "data",
process_jobname = "data",
input_level_string = "data",

-- node list processing
contribute_filter = "simple",
buildpage_filter = "simple",
build_page_insert = "exclusive",
pre_linebreak_filter = "list",
linebreak_filter = "exclusive",
append_to_vlist_filter = "exclusive",
post_linebreak_filter = "reverselist",
hpack_filter = "list",
vpack_filter = "list",
hpack_quality = "list",
vpack_quality = "list",
process_rule = "exclusive",
pre_output_filter = "list",
hyphenate = "simple",
ligaturing = "simple",
kerning = "simple",
insert_local_par = "simple",
mlist_to_hlist = "exclusive",

-- information reporting
pre_dump = "simple",
start_run = "simple",
stop_run = "simple",
start_page_number = "simple",
stop_page_number = "simple",
show_error_hook = "simple",
show_error_message = "simple",
show_lua_error_hook = "simple",
start_file = "simple",
call_edit = "simple",
finish_synctex = "simple",
wrapup_run = "simple",

-- pdf related
finish_pdffile = "data",
finish_pdfpage = "data",
page_order_index = "data",
process_pdf_image_content = "data",

-- font related
define_font = "exclusive",
glyph_not_found = "exclusive",
glyph_info = "exclusive",

-- undocumented
glyph_stream_provider = "exclusive",
provide_charproc_data = "exclusive",
 }

Return a list containing descriptions of added callback functions for specific callback.

function callback.callback_descriptions(name)
    return callback_description[name] or {}
end

local valid_callback_types = {
exclusive = true,
simple = true,
data = true,
}
Create a user callback that can only be called manually using call_callback. A default function is only needed by "exclusive" callbacks.

```lua
function callback.create_callback(name, cbtype, default)
    if callback_types[name] then
        err("cannot create callback ".name." - it already exists")
    elseif not valid_callback_types[cbtype] then
        err("cannot create callback ".name. " with invalid callback type ".cbtype.""")
    elseif cbtype == "exclusive" and not default then
        err("unable to create exclusive callback ".name.", default function is required")
    end

    callback_types[name] = cbtype
    default_functions[name] = default or nil
    user_callbacks[name] = true
end
```

Add a function to the list of functions executed when callback is called. For standard luatex callback a proxy function that calls our machinery is registered as the real callback function. This doesn’t happen for user callbacks, that are called manually by user using call_callback or for standard callbacks that have default functions – like mlist_to_hlist (see below).

```lua
local call_callback

function callback.add_to_callback(name, fn, description)
    if user_callbacks[name] or callback_functions[name] or default_functions[name] then
        -- either:
        -- a) user callback - no need to register anything
        -- b) standard callback that has already been registered
        -- c) standard callback with default function registered separately
        -- (mlist_to_hlist)
    elseif callback_types[name] then
        -- This is a standard luatex callback with first function being added,
        -- register a proxy function as a real callback. Assert, so we know
        -- when things break, like when callbacks get redefined by future
        -- luatex.
        callback_register(name, function(...)
                        return call_callback(name, ...)
                    end)
    else
        err("cannot add to callback ".name." - no such callback exists")
    end

    -- add function to callback list for this callback
    callback_functions[name] = callback_functions[name] or {}
    table.insert(callback_functions[name], fn)

    -- add description to description list
    callback_description[name] = callback_description[name] or {}
    table.insert(callback_description[name], description)
end
```

Remove a function from the list of functions executed when callback is called. If last function in the list is removed delete the list entirely.

```lua
function callback.remove_from_callback(name, description)
    local descriptions = callback_description[name]
    local index
    for i, desc in ipairs(descriptions) do
        if desc == description then
            index = i
            break
        end
    end
    table.remove(descriptions, index)
    local fn = table.remove(callback_functions[name], index)
end
```
if #descriptions == 0 then
    -- Delete the list entirely to allow easy checking of "truthiness".
callback_functions[name] = nil
if not user_callbacks[name] and not default_functions[name] then
    -- this is a standard callback with no added functions and no
    -- default function (i.e. not mlist_to_hlist), restore standard
    -- behaviour by unregistering.
callback_register(name, nil)
end
return fn, description

helper iterator generator for iterating over reverselist callback functions

local function reverse_ipairs(t)
    local i, n = #t + 1, 1
    return function()
        i = i - 1
        if i >= n then
            return i, t[i]
        end
    end
end

Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    -- either take added functions or the default function if there is one
    local functions = callback_functions[name] or {default_functions[name]}
    if cbtype == nil then
        err("cannot call callback '"..name.."' - no such callback exists")
    elseif cbtype == "exclusive" then
        -- only one function, atleast default function is guaranteed by
        -- create_callback
        return functions[1](...)
    elseif cbtype == "simple" then
        -- call all functions one after another, no passing of data
        for _, fn in ipairs(functions) do
            fn(...) end
        return
    elseif cbtype == "data" then
        -- pass data (first argument) from one function to other, while keeping
        -- other arguments
        local data = ...
        for _, fn in ipairs(functions) do
            data = fn(data, select(2, ...))
        end
        return data
    end
    -- list and reverselist are like data, but "true" keeps data (head node)
    -- unchanged and "false" ends the chain immediately
    local iter
    if cbtype == "list" then
        iter = ipairs
    elseif cbtype == "reverselist" then
        iter = reverse_ipairs
    end
local head = (...)  
local new_head = ...
local changed = false
for _, fn in ipairs(functions) do
    new_head = fn(head, select(2, ...))
    if new_head == false then
        return false
    elseif new_head ~= true then
        head = new_head
        changed = true
    end
end
return not changed or head
end

call_callback = callback.call_callback
Create “virtual” callbacks pre/post_mlist_to_hlist_filter by setting mlist_to_hlist callback. The default behaviour of mlist_to_hlist is kept by using a default function, but it can still be overriden by using add_to_callback.

default_functions["mlist_to_hlist"] = node.mlist_to_hlist
callback.create_callback("pre_mlist_to_hlist_filter", "list")
callback.create_callback("post_mlist_to_hlist_filter", "reverselist")
callback_register("mlist_to_hlist", function(head, ...)  
-- pre_mlist_to_hlist_filter
local new_head = call_callback("pre_mlist_to_hlist_filter", head, ...)  
if new_head == false then
    node.flush_list(head)
    return nil
elseif new_head ~= true then
    head = new_head
end
-- mlist_to_hlist means either added functions or standard latex behavior
-- of node.mlist_to_hlist (handled by default function)
head = call_callback("mlist_to_hlist", head, ...)  
-- post_mlist_to_hlist_filter
new_head = call_callback("post_mlist_to_hlist_filter", head, ...)  
if new_head == false then
    node.flush_list(head)
    return nil
elseif new_head ~= true then
    head = new_head
end
return head
end)

For preprocessing boxes just before shipout we define custom callback. This is used for coloring based on attributes. There is however a challenge - how to call this callback? We could redefine \shipout and \pdfxform (which both run ship_out procedure internally), but they would lose their primitive meaning – i.e. \immediate wouldn’t work with \pdfxform. The compromise is to require anyone to run \_preshipout⟨destination box number⟩⟨box specification⟩ just before \shipout or \pdfxform if they want to call pre_shipout_filter (and achieve colors and possibly more).

callback.create_callback("pre_shipout_filter", "list")
local tex_setbox = tex.setbox
local token_scanint = token.scan_int
local token_scanlist = token.scan_list
define_lua_command("_preshipout", function()
    local boxnum = token_scanint()
    local head = token_scanlist()
    head = call_callback("pre_shipout_filter", head)
    tex_setbox(boxnum, head)
end)

Compatibility with \TeX through luatexbase namespace. Needed for luaotfload.

luatexbase = {

391       register_number = register_number,
392       attributes = attributes,
393       provides_module = provides_module,
394       new_attribute = alloc.new_attribute,
395       callback_descriptions = callback.callback_descriptions,
396       create_callback = callback.create_callback,
397       add_to_callback = callback.add_to_callback,
398       remove_from_callback = callback.remove_from_callback,
399       call_callback = callback.call_callback,
400       callbacktypes = {} }
401 }

\tracingmacros callback registered. Use \tracingmacros=3 or \tracingmacros=4 if you want to see the result.

405       callback.add_to_callback("input_level_string", function(n)
406         if tex.tracingmacros > 3 then
407           return \[
408             .. n .. \]
409         elseif tex.tracingmacros > 2 then
410           return ~ .. string.rep(\.,n)
411         else
412           return 
413         end
414       end, ".tracingmacros")

2.39.4 Handling of colors using attributes

Because LuaTeX doesn’t do anything with attributes, we have to add meaning to them. We do this by
intercepting \TeX{} just before it ships out a page and inject PDF literals according to attributes.

421       local node_id = node.id
422       local node_subtype = node.subtype
423       local glyph_id = node_id("glyph")
424       local rule_id = node_id("rule")
425       local glue_id = node_id("glue")
426       local hlist_id = node_id("hlist")
427       local vlist_id = node_id("vlist")
428       local disc_id = node_id("disc")
429       local whatsit_id = node_id("whatsit")
430       local pdfliteral_id = node_subtype("pdf_literal")
431       local pdfsave_id = node_subtype("pdf_save")
432       local pdfrestore_id = node_subtype("pdf_restore")
433       local token_getmacro = token.get_macro
434
435       local direct = node.direct
436       local todirect = direct.todirect
437       local tonode = direct.tonode
438       local getfield = direct.getfield
439       local setfield = direct.setfield
440       local getwhd = direct.getwhd
441       local getid = direct.getid
442       local getlist = direct.getlist
443       local setlist = direct.setlist
444       local getleader = direct.getleader
445       local getattribute = direct.get_attribute
446       local insertbefore = direct.insert_before
447       local copy = direct.copy
448       local traverse = direct.traverse
449       local one_bp = tex.sp("1bp")
450       local string_format = string.format

The attribute for coloring is allocated in \colors_opm

453       local color_attribute = register_number("_colorattr")

Now we define function which creates \whatsit nodes with PDF literals. We do this by creating a
base literal, which we then copy and customize.

458       local pdf_base_literal = direct.new("whatsit", "pdf_literal")
459       setfield(pdf_base_literal, "node", 2) -- direct node
The function `colorize(head, current, current_stroke)` goes through a node list and injects PDF literals according to attributes. Its arguments are the head of the list to be colored and the current color for fills and strokes. It is a recursive function – nested horizontal and vertical lists are handled in the same way. Only the attributes of “content” nodes (glyphs, rules, etc.) matter. Users drawing with PDF literals have to set color themselves.

Whatsit node with color setting PDF literal is injected only when a different color is needed. Our injection does not care about boxing levels, but this isn’t a problem, since PDF literal whatsits just instruct the `\shipout` related procedures to emit the literal.

We also set the stroke and non-stroke colors separately. This is because stroke color is not always needed – LuaTEX itself only uses it for rules whose one dimension is less than or equal to 1 bp and for fonts whose `mode` is set to 1 (outline) or 2 (outline and fill). Catching these cases is a little bit involved. For example rules are problematic, because at this point their dimensions can still be running (\(\sim 2^{30}\)) – they may or may not be below the one big point limit. Also the text direction is involved. Because of the negative value for running dimensions the simplistic check, while not fully correct, should produce the right results. We currently don’t check for the font mode at all.

Leaders (represented by glue nodes with leader field) are not handled fully. They are problematic, because their content is repeated more times and it would have to be ensured that the coloring would be right even for e.g. leaders that start and end on a different color. We came to conclusion that this is not worth, hence leaders are handled just opaquely and only the attribute of the glue node itself is checked. For setting different colors inside leaders, raw PDF literals have to be used.

We use the `node.direct` way of working with nodes. This is less safe, and certainly not idiomatic Lua, but faster and codewise more close to the way TeX works with nodes.

```lua
local function pdfliteral(str)
    local literal = copy(pdf_base_literal)
    setfield(literal, "data", str)
    return literal
end
optex.directpdfliteral = pdfliteral

local function is_color_needed(head, n, id, subtype) -- returns non-stroke, stroke color needed
    if id == glyph_id then
        return true, false
    elseif id == glue_id then
        n = getleader(n)
        if n then
            id = getid(n)
            if id == hlist_id or id == vlist_id then
                -- leaders with hlist/vlist get single color
                return true, false
            else -- rule
                -- stretchy leaders with rules are tricky,
                -- just set both colors for safety
                return true, true
            end
        end
    elseif id == rule_id then
        local width, height, depth = getwhd(n)
        if width <= one_bp or height + depth <= one_bp then
            -- running (\(\sim 2^{30}\)) may need both
            return true, true
        end
    elseif id == whatsit_id and (subtype == pdfliteral_id
        or subtype == pdfsave_id
        or subtype == pdfrestore_id) then
        return true, true
    end
    return false, false
end

local function colorize(head, current, current_stroke)
    for n, id, subtype in traverse(head) do
        if id == hlist_id or id == vlist_id then
            -- nested list, just recurse
        end
        local list = getlist(n)
```
list, current, current_stroke = colorize(list, current, current_stroke)
setlist(n, list)
elseif id == disc_id then
  -- at this point only no-break (replace) list is of any interest
  local replace = getfield(n, "replace")
  if replace then
    replace, current, current_stroke = colorize(replace, current, current_stroke)
    setfield(n, "replace", replace)
  end
  else
    local nonstroke_needed, stroke_needed = is_color_needed(head, n, id, subtype)
    local new = getattribute(n, color_attribute) or 0
    local newcolor = nil
    if current ~= new and nonstroke_needed then
      newcolor = token_getmacro("_color:"..new)
      current = new
    end
    if current_stroke ~= new and stroke_needed then
      local stroke_color = token_getmacro("_color-s:"..current)
      if newcolor then
        newcolor = string_format("%s %s", newcolor, stroke_color)
      else
        newcolor = stroke_color
      end
      current_stroke = new
    end
  end
  if newcolor then
    head = insertbefore(head, n, pdfliteral(newcolor))
end
end
return head, current, current_stroke
end

Colorization should be run just before shipout. We use our custom callback for this. See the definition of pre_shipout_filter for details on limitations.

callback.add_to_callback("pre_shipout_filter", function(list)
  -- By setting initial color to -1 we force initial setting of color on
  -- every page. This is useful for transparently supporting other default
  -- colors than black (although it has a price for each normal document).
  local list = colorize(todirect(list), -1, -1)
  return tonode(list)
end, ".colors")

We also hook into luaotfload’s handling of color. Instead of the default behavior (inserting colorstack whatsts) we set our own attribute. The hook has to be registered after luaotfload is loaded.

function optex_hook_into_luaotfload()
  if not luaotfload.set_colorhandler then
    return -- old luaotfload, colored fonts will be broken
  end
  local setattribute = direct.set_attribute
  local token_setmacro = token.set_macro
  local color_count = registernumber("_colorcnt")
  local tex_getcount, tex_setcount = tex.getcount, tex.setcount
  luaotfload.set_colorhandler(function(head, n, rgbcolor) -- rgbcolor = "1 0 0 rg"
    local attr = tonumber(token_getmacro("_color::"..rgbcolor))
    if not attr then
      attr = tex_getcount(color_count)
    tex_setcount(color_count, attr + 1)
    local setattr = tostring(attr)
    token_setmacro("_color::"..rgbcolor, setattr)
    token_setmacro("_color:"..strattr, rgbcolor)
  end
  end
  setattribute(n, color_attribute, attr)
  return head, n
end
2.40 Printing documentation

The \printdoc {filename}{space} and \printdoctail {filename}{space} commands are defined after the file \file{doc.opm} is loaded by \load [doc].

The \printdoc command starts reading of given \( {filename} \) from the second line. The file is read in the listing mode. The \printdoctail command starts reading given \( {filename} \) from the first occurrence of the \_endcode. The file is read in normal mode (like \input {filename}).

The listing mode prints the lines as a listing of a code. This mode is finished when first \_doc occurs or first \_endcode occurs. At least two spaces or one tab character must precede before such \_doc. On the other hand, the \_encode must be at the left edge of the line without spaces. If this rule is not met then the listing mode continues.

If the first line or the last line of the listing mode is empty then such lines are not printed. The maximal number of printed lines in the listing mode is \maxlines. It is set to almost infinity (100000). You can set it to a more sensible value. Such a setting is valid only for the first following listing mode.

When the listing mode is finished by \_doc then the next lines are read in the normal way, but the material between \_begtt \_endtt pair is shifted by three letters left. The reason is that the three spaces of indentation is recommended in the \_doc ... \_cod pair and this shifting is compensation for this indentation.

The \_cod macro ignores the rest of the current line and starts the listing mode again.

When the listing mode is finished by \_endcode then the \endinput command is applied, the reading of the file opened by \printdoc is finished.

You cannot reach the end of the file (without \_endcode) in the listing mode.

The main documentation point is denoted by \_\{\sequence\} in red, for example \_\{foo\}. The user documentation point is the first occurrence of \_\{\sequence\}, for example \_\{foo\}. There can be more such markups, all of them are hyperlinks to the main documentation point. And main documentation point is a hyperlink to the user documentation point if this point precedes. Finally, the \~\{\sequence\} (for example \~\{foo\}) are hyperlinks to the user documentation point.

By default, the hyperlink from main documentation point to the user documentation point is active only if it is backward link, i.e. the main documentation point is given later. The reason is that we don’t know if such user documentation point will exist when creating main documentation point and we don’t want broken links. If you are sure that user documentation point will follow then use prefix \fw before \_, for example \fw\_\{foo\} is main documentation point where the user documentation point is given later and forward hyperlink is created here.

Control sequences and their page positions of main documentation points and user documentation points are saved to the index.

The listing mode creates all control sequences which are listed in the index as an active link to the main documentation point of such control sequence and prints them in blue. Moreover, active links are control sequences of the type \_\_foo or \_foo although the documentation mentions only \_foo. Another text is printed in black.

The listing mode is able to generate external links to another OpTEX-like documentation, if the macros \_\{csname\} and \el:\{csname\} are defined. The second macro should create a hyperlink using \_tmpa where the link name of the \{csname\} is saved and \_tmpb where the name of the \{csname\} to be printed is saved (\_tmpb can include preceding _ or . unlike \_tmpa). For example, suppose, that we have created optex-doc.eref file by:

```
TEXINPUTS='.;$TEXMF/{doc,tex}://' optex optex-doc
grep Xindex optex-doc.ref > optex-doc.eref
```

The .eref file includes only \_Xindex{\{csname\}}{} lines from optex-doc.ref file. Then we can use following macros:
All \csname curname \endcsname, where \csname curname \endcsname is from optex-doc.ref, have the same meaning: \texttt{optexdoclink} in this example. And \texttt{optexdoclink} creates the external link in \texttt{Cyan} color.

### 2.40.1 Implementation

```latex
\newcount \maxlines \maxlines = 100000
\public \maxlines ;
\edef \cod#1{\par \wipepar}
\edef \readverbline #1\linebreak{\def \tmpa{\empty#1} \let \next=\readverbline}
\edef \Endcode{\def \tmpb{\empty}}
\edef \catcodedot{\catcode `. = 11}
```

The listing mode is implemented here. The \texttt{\maxlines} is maximal lines of code printed in the listing mode. The \texttt{\catcodedot} sets dot as letter in listngs (for package documentation where \texttt{.foo} sequeces exist).
The scanner of the control sequences in the listing mode replaces all occurrences of \ by \_makecs. This macro reads next tokens and accumulates them to \_tmpa as long as they have category 11. It means that \_tmpa includes the name of the following control sequence when \_makecsF is run. The printing form of the control sequence is set to \_tmpb and the test of existence \langle csname\rangle is performed. If it is true then active hyperlink is created. If not, then the first \_ or . is removed from \_tmpa and the test is repeated.

\begin{verbatim}
\def\makecs{%\def\tmpa{}\futurelet\nextmakecsA}
\def\makecsA{%\ifcat a\noexpand\next \ea\makecsB \else \ea\makecsF \fi}
\def\makecsB#1{%\addto\tmpa{#1}\futurelet\nextmakecsA}
\def\makecsF{%\let\tmpb=\tmpa
  \ifx\tmpa\empty\csstring\%
  \else \ifcsname,\tmpa\endcsname\trycs{el:\tmpa}{\intlink}\%
  \else\remfirstunderscoreordot\tmpa
  \ifx\tmpa\empty\let\tmpa=\tmpb\fi\fi\fi\fi\fi
}
\def\processinput{%\let\start=\relax\ea\replstring\ea\tmpb\ea{\aspace\^^J}{\^^J}
  \addto\tmpb{\end}\isinlist\tmpb{\start\^^J}\iftrue \advance\ttline by1 \fi
  \replstring\tmpb{\start\^^J}{\start}\replstring\tmpb{\start}{\relax}
  \replstring\tmpb{\^^J\end}{\end}\replstring\tmpb{\^^J\end}{\relax}
  \replstring\tmpb{\end}{\relax}\ea\prepareverbdata\ea\tmpb\ea{\tmpb\^^J}\%}
\def\remfirstunderscoreordot#1{\ea\remfirstuordotA#1\relax#1}
\def\remfirstuordotA#1#2\relax#3{%\if _#1\def#3{#2}\fi\if\string#1.\def#3{#2}\fi}
\end{verbatim}

By default the internal link is created by \_intlink inside listing mode. But you can define \el:\langle csname\rangle which has precedence and it can create an external link. The \_tmpa includes the name used in the link and \_tmpb is the name to be printed. See \_makecsF above and the example at the beginning of this section.

\begin{verbatim}
\def\intlink{%\link[cs:\tmpa]{\Blue}{\csstring\tmpb}}
\end{verbatim}

The lines in the listing mode have a yellow background.

\begin{verbatim}
\def\Yellow{%\setcmykcolor{0 0 .3 .03}}
\def\printcodeline#1{%\advance\maxlines by-1
  \ifnum\maxlines<0\ea\endverbprinting \fi
  \if\printfilename\relax\penalty\ttpenalty\fi\vskip-4pt
  \noindent\rlap\Yellow\vrule height8pt depth5pt width\hsize\%\printfilename\indent\printverblinenum #1\par}
\def\printfilename{%\hbox to0pt{\hskip\hsize\vbox to0pt{\vss\llap{\Brown\docfile}\kern7.5pt}}\hss}
\let\printfilename=\relax
\everytt={\let\printverblinenum=\relax}
\long\def\endverbprinting{%\end\fi\global\maxlines=100000
  \noindent\typosize[8/]{\dots etc. (see \tt\Brown\docfile)}}
\end{verbatim}

\docfile is currently documented file. \printdoc and \printdoctail macros are defined here.
You can do `\verbinput` \vitt}\{<filename>\} \{(from)-(to)\} <filename> if you need analogical design like in listing mode.

The Index entries are without the trailing backslash. We must add it when printing Index.

If this macro is loaded by `\load` then we need to initialize catcodes using the `\_afteroad` macro.

The <something> will be print as ⟨something⟩.

Main documentation points and hyperlinks to/from it. Main documentation point: `\`foo`. User documentation point: `~`foo, first occurrence only. The next occurrences are only links to the main documentation point. Link to user documentation point: `~`foo.
The \texttt{fw} macro for forward links to user documentation point (given later) is defined here.

\begin{verbatim}
\texttt{fw} \texttt{\ldef\`#1`{{\slet{cs:`\csstring#1}{}\`#1`}}
\public \texttt{fw} ;
\end{verbatim}
Index
There are all control sequences used in \texttt{Op\TeX} except \texttt{TeX} primitives. If you want to know something about \texttt{TeX} primitives then you can use another index from \texttt{TeX} in a Nutshell.

\_above\textit{el}skip 128
\_above\textit{v}etitle 123, 126
\activequotes 189
\addcititelist 153
\addcolor 111
\addextgstate 142
\ad\textit{d}itcorr 103
\add\textit{d}dress 25, 178–179
\add\textit{d}tabl\textit{em}x 147
\add\textit{d}to 28, 38, 54, 104
\add\textit{d}tomodlist 76
\ad\textit{f} 17, 28, 38
\ad\textit{dots} 86
\advance\textit{p}ageno 104–105
\afterfi 28, 41
\_after\textit{itc}orr 103
\after\textit{load} 52
\_alloc\textit{ator} 40
\allow\textit{break} 56
\alt\textit{quotes} 188–189
\_asciisort\textit{ing}true 173
\_at\textit{he} 130
\_auth\textit{urname} 156
\b 58
\_background\textit{box} 105
\background\textit{pic} 140
\bb\textit{char} 80, 95
\_beg\textit{block} 14, 27, 130
\_beg\textit{t}items 13–14, 27, 48, 129
\beg\textit{multi} 19, 27, 48, 150
\_beg\textit{output} 104–105, 120
\beg\textit{tt} 16–18, 27, 47–48, 105, 131, 133
\_beg\textit{tti} 131
\below\textit{w}iskip 128
\_be\textit{low}title 123, 126
\bf 8–9, 63–64, 73, 80, 95
\bg\textit{roup} 37
\bi 8–9, 63–64, 73, 80, 95
\bib 20–21, 27, 154
\_bi\textit{bA} 154
\_bi\textit{BB} 154
\_bi\textit{bgl} 154
\_bi\textit{mark} 152, 154, 156
\_bi\textit{bmn} 153
\_bi\textit{bnum} 116, 152
\_bi\textit{bop}tions 49, 161
\_bi\textit{bp} 152
\_bi\textit{bpart} 21, 49, 152
\_bi\textit{bskip} 155
\_bi\textit{tex}hook 48, 155
\_bi\textit{bwarning} 156, 159
\big 85
\_Big 85
\_big\textit{break} 56
\_bigg 85
\_biggl 85
\_biggr 85
\_bigm 85
\_bigmm 85
\_bign 85
\_bign 85
\_bigr 85
\_bigsl 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85
\_bigr 85

\_centerline 57
\chap 10, 12, 17–18, 27, 53, 123, 125
\_chap\textit{font} 66, 123
\chapx 124
\check\textit{exists} 34
\chyp 24, 190
\_c\textit{ircle} 140–141
\circle\textit{params} 50
\_cite 12, 20–21, 27, 152, 155
\_cite\textit{A} 152
\_cite\textit{E} 130
\_cite\textit{I} 153
\_clip\textit{incircle} 24, 143
\clip\textit{oval} 24, 143
\clip\textit{in\textit{path}} 143
\cl\textit{q} 190
\_cm\textit{ykcolor\textit{def}} 111
\_cm\textit{yktor\textit{gb}} 110–111
\_cn\textit{vinfo} 51
\_c\textit{ode} 28, 33–34, 54
\_c\textit{ode} 16–17, 27, 47, 130
\_c\textit{ode\textit{cl}} 28, 33–35
\_c\textit{ol\textit{num}} 147
\_c\textit{olor\textit{attr}} 108, 110
\_c\textit{olor\textit{cn}} 110
\_c\textit{olor\textit{crop}} 110
\_c\textit{olor\textit{def}} 22, 28, 108–110, 112
\_c\textit{olor\textit{def\textit{Fin}}} 110
\_colorp\textit{refix} 110
\_col\textit{sep} 48
\_com\textit{ment\textit{chars}} 18, 132–134
\_comm\textit{oncolor\textit{def}} 111
\_com\textit{plete\textit{page}} 104–105
\_com\textit{pound\textit{chars}} 171
\_com\textit{pound\textit{chars\textit{sc}}} 171
\_com\textit{pound\textit{chars\textit{sen}}} 171
\_con\textit{g} 88
\_corr\textit{msize} 81, 93
\_c\textit{ramp\textit{ed}} 90
\_c\textit{rl} 15, 146, 149
\_c\textit{rli} 15, 144, 147, 149
\_c\textit{rll} 15, 149
\_c\textit{rll\textit{i}} 15, 144, 149
\_c\textit{rlp} 15, 144, 149
\_cr\textit{cq} 190
\_c\textit{s} 28, 38
\_c\textit{S} 183
\_c\textit{s\textit{k}}ip 10, 127
\_c\textit{s\textit{lang}} 24, 184
\_c\textit{plain} 183
\_c\textit{quotes} 25, 188

206