OpTeX

Format Based on Plain \TeX{} and OPmac\footnote{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 \url{http://petr.olsak.net/opmac-e.html} for more information about it.}

Petr Olšák, 2020, 2021, 2022, 2023

Verison 1.13

http://petr.olsak.net/optex

OpTeX is Lua\TeX{} format with Plain \TeX{} and OPmac. Only Lua\TeX{} 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\footnote{All these features are implemented by \TeX{} macros, no external program is needed.}, reading \texttt{.bib} files directly\footnote{All these features are implemented by \TeX{} macros, no external program is needed.}, syntax highlighting\footnote{All these features are implemented by \TeX{} macros, no external program is needed.}, 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 ($\texttt{\foo} \text{ versus } \texttt{\_foo}$). 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.
Contents

1 User documentation .......................................................... 5
  1.1 Starting with Op\TeX ......................................................... 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, transparency ............................................... 21
    1.6.2 Images .................................................................. 22
    1.6.3 PDF transformations ............................................... 23
    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 ................................. 26
    1.7.4 Lorem ipsum dolor sit ............................................ 26
    1.7.5 Logos .................................................................... 26
    1.7.6 The last page .......................................................... 27
    1.7.7 Use Op\TeX ............................................................ 27
    1.7.8 Op\TeX tricks ......................................................... 27
  1.8 Summary ...................................................................... 28
  1.9 API for macro writers ...................................................... 29
  1.10 Compatibility with Plain \TeX .............................................. 30
  1.11 Related documents .......................................................... 30

2 Technical documentation ....................................................... 31
  2.1 The main initialization file ............................................... 31
  2.2 Basic principles of Op\TeX sources ..................................... 33
    2.2.1 Concept of namespaces of control sequences .......... 33
    2.2.2 Macro files syntax ............................................... 34
    2.2.3 Name spaces for package writers ......................... 34
    2.2.4 Summary about rules for external macro files published for Op\TeX 35
2.2.5 The implementation of the namespaces and macros for macro-files

2.3 \pdf\TeX\ initialization

2.4 Basic macros

2.5 If-macros, loops, is-macros

2.6 Allocators for \TeX\ registers

2.7 Setting parameters

2.8 More \op\TeX\ macros

2.9 Using key=value format in parameters

2.10 Plain \TeX\ macros

2.11 Preloaded fonts for text mode

2.12 Using \font\ primitive directly

2.13 The Font Selection System

2.14 Preloaded fonts for math mode

2.15 Math macros

2.16 Unicode-math fonts

2.17 Scaling fonts in document (high-level macros)

2.18 Output routine

2.19 Margins

2.20 Colors

2.20.1 Basic concept

2.20.2 Color mixing
2.20.3 Implementation ................................................. 118
2.21 The .ref file ..................................................... 123
2.22 References ........................................................ 124
2.23 Hyperlinks ........................................................ 126
2.24 Making table of contents ........................................ 128
2.25 PDF outlines ...................................................... 130
  2.25.1 Nesting PDF outlines ....................................... 130
  2.25.2 Strings in PDF outlines ................................... 131
2.26 Chapters, sections, subsections .................................. 133
2.27 Lists, items ....................................................... 138
2.28 Verbatim, listings ................................................ 140
  2.28.1 Inline and “display” verbatim ............................. 140
  2.28.2 Listings with syntax highlighting ......................... 145
2.29 Graphics .......................................................... 148
2.30 The \table macro, tables and rules .............................. 154
  2.30.1 The boundary declarator : ................................ 154
  2.30.2 Usage of the \tabskip primitive ............................ 154
  2.30.3 Tables to given width ...................................... 154
  2.30.4 \eqbox: boxes with equal width across the whole document . 155
  2.30.5 Implementation of the \table macro and friends .......... 155
2.31 Balanced multi-columns .......................................... 160
2.32 Citations, bibliography .......................................... 162
  2.32.1 Macros for citations and bibliography preloaded in the format . 162
  2.32.2 The \usebib command ....................................... 166
  2.32.3 Notes for bib-style writers ................................ 166
  2.32.4 Direct reading of .bib files ................................. 168
  2.32.5 The usebib.omp macro file loaded when \usebib is used ... 168
  2.32.6 Usage of the bib-iso690 style ............................. 174
  2.32.7 Implementation of the bib-iso690 style ................. 181
2.33 Sorting and making Index ........................................ 185
2.34 Footnotes and marginal notes .................................... 192
2.35 Styles ............................................................ 194
  2.35.1 \report and \letter styles ................................. 194
  2.35.2 \slides style for presentations ............................ 195
2.36 Logos .............................................................. 199
2.37 Multilingual support .............................................. 199
  2.37.1 Lowercase, uppercase codes ................................ 199
  2.37.2 Multilingual phrases and quotation marks ............... 200
  2.37.3 Languages declaration .................................... 202
  2.37.4 Data for various languages ................................ 205
2.38 Other macros ...................................................... 207
2.39 Lua code embedded to the format ................................ 210
  2.39.1 General ....................................................... 210
  2.39.2 Allocators .................................................... 210
  2.39.3 Callbacks ..................................................... 211
  2.39.4 Management of PDF page resources ....................... 216
  2.39.5 Handling of colors and transparency using attributes ... 217
2.40 Printing documentation ........................................... 220
  2.40.1 Implementation .............................................. 221

Index 225
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

    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 http://petr.olsak.net/optex.

    A minimal document should be

    \fontfam[LMfonts]
    Hello World! \bye

    The first line \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\(^1\). So the sentence Hello World will be OK without the first line, but you cannot print such sentence in other languages (for example Ahoj světe!) where Unicode fonts are needed because the characters like ě are not mapped correctly in preloaded fonts.

    A somewhat larger example with common settings should be:

    \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)

    Tady je zkušební textík v českém jazyce.
    \bye

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

1.2 Page layout

1.2.1 Setting the margins

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

    \margins/⟨pg⟩ ⟨fmt⟩ ⟨⟨left⟩,⟨right⟩,⟨top⟩,⟨bot⟩⟩⟨unit⟩

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

    Parameters are:
    • ⟨pg⟩ ... 1 or 2 specifies one-page or two-pages design.
    • ⟨fmt⟩ ... paper format (a4, a4l, a5, letter, etc. or user defined).
    • ⟨left⟩, ⟨right⟩, ⟨top⟩, ⟨bot⟩ ... gives the amount of left, right, top and bottom margins.
    • ⟨unit⟩ ... unit used for values ⟨left⟩, ⟨right⟩, ⟨top⟩, ⟨bot⟩.

\(^1\) This is a technical limitation of LuaTeX for fonts downloaded in formats: only 8bit fonts can be preloaded.
Each of the parameters \(\langle left\rangle\), \(\langle right\rangle\), \(\langle top\rangle\), \(\langle bot\rangle\) can be empty. If both \(\langle left\rangle\) and \(\langle right\rangle\) are nonempty then \(\text{\textbackslash hsize}\) is set. Else \(\text{\textbackslash hsize}\) is unchanged. If both \(\langle left\rangle\) and \(\langle right\rangle\) are empty then typesetting area is centered in the paper format. The analogical rule works when \(\langle top\rangle\) or \(\langle bot\rangle\) parameter is empty (\(\text{\textbackslash vsize}\) instead \(\text{\textbackslash hsize}\) is used). Examples:

\begin{verbatim}
\margins/1 a4 (,,,)mm % \textbackslash hsize, \textbackslash vsize untouched,
% typesetting area centered
\margins/1 a4 (,2,,)cm % right margin set to 2cm
% \textbackslash hsize, \textbackslash vsize untouched, vertically centered
\end{verbatim}

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

Op\TeX\ declares following paper formats: a4, a4l (landscape a4), a5, a5l, a3, a3l, b5, letter and user can declare another own format by \texttt{\textbackslash sdef}:

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

The \(\langle fmt\rangle\) can be also in the form \((\langle width\rangle,\langle height\rangle)\langle unit\rangle\) where \(\langle unit\rangle\) is optional. If it is missing then \(\langle unit\rangle\) after margins specification is used. For example:

\begin{verbatim}
\margins/1 (100,200) (7,7,7,7)mm
declares the paper 100×200mm with all four margins 7mm. The spaces before and after \(\langle fmt\rangle\) parameter are necessary.
\end{verbatim}

The command \texttt{\textbackslash magscale}[\langle factor\rangle] 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:

\begin{verbatim}
\margins/2 a5 (22,17,19,21)mm
\magscale[1414] \margins/1 a4 (,,,)mm
\end{verbatim}

The first line sets the \textbackslash hsize and \textbackslash 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

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

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

The distance between the \texttt{\textbackslash headline} and the top of the page body is given by the \texttt{\textbackslash headedist} register. The distance between bottom of the page body and the \texttt{\textbackslash footline} is given by \texttt{\textbackslash footlinedist}. The default values are:

\begin{verbatim}
\headline = {} \\
\footline = {\\textbackslash hss\textbackslash rmfixed \_folio \_hss} \% \textbackslash folio expands to page number \\
\headlinedist = 14pt \% from baseline of \textbackslash headline to top of page body \\
\footlinedist = 24pt \% from last line in pagebody to baseline of footline
\end{verbatim}
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{\textbackslash{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{\textbackslash{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.

\subsection{Footnotes and marginal notes}

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

The \texttt{\textbackslash{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{\textbackslash{fnotemark}}\{\langle numeric-label\rangle\} inside the box: only the footnote mark is generated here. When the box is finished you can use \texttt{\textbackslash{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 \texttt{\textbackslash{fnotemark}} inside the same box has to have the parameter 2 etc. The same number of \texttt{\textbackslash{fnotetext}}s have to be written after the box as the number of \texttt{\textbackslash{fnotemarks}} inserted inside the box.

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

The marginal note can be printed by the \texttt{\textbackslash{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{\textbackslash{fixmnotes}right} or \texttt{\textbackslash{fixmnotes}left}.

The \langle text\rangle is formatted as a little paragraph with the maximal width \texttt{\textbackslash{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{\textbackslash{mnote}} in the text. The exceptions are possible by using the \texttt{up} keyword: \texttt{\textbackslash{mnote up}}\{\langle dimen\rangle\}\{\langle text\rangle\}. You can set each \langle dimen\rangle to each \texttt{\textbackslash{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{\textbackslash{mnote up}2\textbackslash{baselineskip}}\{\langle text\rangle\} shifts this marginal note two lines up.

\subsection{Fonts}

\subsubsection{Font families}

You can select the font family by \texttt{\textbackslash{fontfam}}\{(\langle Family-name\rangle)\}. The argument \langle Family-name\rangle is case insensitive and spaces are ignored in it. For example, \texttt{\textbackslash{fontfam}}[\texttt{LM Fonts}] is equal to \texttt{\textbackslash{fontfam}}[\texttt{LMFonts}] and it is equal to \texttt{\textbackslash{fontfam}}[\texttt{lmfonts}]. Several aliases are prepared, thus \texttt{\textbackslash{fontfam}}[\texttt{Latin Modern}] can be used for loading Latin Modern family too.
If you write \fontfam[?] then all font families registered in OpTEX are listed on the terminal and in the log file. If you write \fontfam[catalog] then a catalog of all fonts registered in OpTEX and available in your TEX system is printed. See also this catalog.

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

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

Example:

This is \{\texttt{\em important} text. \} = This is \{\texttt{\it important}/\} text.
\it This is \{\texttt{\em important} text. \} = This is\{\texttt{\rm important}\} text.
\bf This is \{\texttt{\em important} text. \} = This is \{\texttt{\bi important}\} text.
\bi This is \{\texttt{\em important} text. \} = This is\{\texttt{\bf important}\} text.

More about the OpTEX Font Selection System is written in the technical documentation in the section 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:

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

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: \(\{\texttt{\typosize[12/14]} \ldots\langle text of paragraph\rangle \ldots \\texttt{\par}\). 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:

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

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{\mainfontsize} and \texttt{\mainbaselineskip}.
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 resize only the current font by the command \texttt{\textbackslash thefontsize}{⟨font-size⟩} or the font can be rescaled by \texttt{\textbackslash thefontscale}{⟨factor⟩}. These macros don’t change math fonts sizes nor baselineskip.

There is “low level” \texttt{\textbackslash setfontsize}{⟨size-spec⟩} 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.

1.3.3 Typesetting math

See the additional document Typesetting Math with OpTEX for more details about this issue.

OpTEX 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}{⟨font-file⟩} or \texttt{\textbackslash loadmath}{⟨font-name⟩} before first \texttt{\textbackslash fontfam} command.

Hundreds math symbols and operators like in AMSTeX are accessible. For example \texttt{\textalpha, \textgeq, \sum \textbackslash sum, \textbackslash sphericalangle \textbackslash \textangle, \textbumpoeq, \textbackslash \textapprox}. See AMSTeX manual or Typesetting Math with OpTEX 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 calligraphics \texttt{ABC–XYZ}
\texttt{\script} \% script \texttt{ABCDEFGHIJKLMNOPQRSTUVWXYZ}
\texttt{\frak} \% fracture \texttt{abc–xyz, ABC–XYZ}
\texttt{\bbchar} \% double stroked letters \texttt{ABC–XYZ}
\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 $\mathbox{\langle text \rangle}$ inside math mode. It behaves as $\hbox{\langle text \rangle}$ (i.e. the $\langle text \rangle$ is printed in horizontal non-math mode) but the size of the $\langle text \rangle$ 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 \langle end of line \rangle
\chap Chapter title \langle end of line \rangle
\sec Section title \langle end of line \rangle
\secc Subsection title \langle end of line \rangle

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 $\LaTeX$ 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 $\p$ at the end of each line (except the last one). When $\p$ 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 $\nl\p$ 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 \quad = \sqrt{a^2+b^2} \quad \& \quad \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:
This example produces:

**Table 1.4.1** The dependency of the computer-dependency on the age.

<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>

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:

```
\topinsert % table and its caption printed at the top of the current page
<caption and table>
\endinsert
```

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\(^3\). They can be used in context of `\numberedpar ⟨letter⟩{⟨text⟩}` macro. For example:

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

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 Let $\text{SM}\$ be...` and the new paragraph is started with the text: **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 **Theorem 1.4.2** (L'Hôpital's rule). Let $f$, $g$ be...

\(^3\) This feature can be changed, see the section 2.26 in the technical documentation.
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 \(\text{\textbackslash chap}, \text{\textbackslash sec}, \text{\textbackslash secc}, \text{\textbackslash caption/t}, \text{\textbackslash caption/f}\) or \(\text{\textbackslash eqmark}\). The alternative syntax is to use \(\text{\textbackslash label}\) before mentioned commands (not necessarily directly before). The reference is realized by \(\text{\textbackslash ref}\) (prints the number of the referenced object) or \(\text{\textbackslash pgref}\) (prints the page number). Example:

\[
\text{\textbackslash sec}[\text{beatle}] \text{ About Beatles}
\]
\[
\text{\textbackslash noindent} \text{\textbackslash hfil} \table[rl]{...} \% \text{the table}
\text{\textbackslash cskip}
\]
\[
\text{\textbackslash label}[\text{pythagoras}] \quad $$$ a^2 + b^2 = c^2 \text{ \textbackslash eqmark} $$$
\]

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 \text{\textbackslash ref} or \text{\textbackslash pgref} can be given explicitly by \text{\textbackslash ref}\{(\text{label})\}\{(\text{text})\} or \text{\textbackslash pgref}\{(\text{label})\}\{(\text{text})\}. If the \(\text{\textbackslash text}\) includes the @ character, it is replaced by implicitly printed text. Example: see \text{\textbackslash ref}\{\text{lab}\}\{\text{section-@}\} prints the same as see section~\text{\textbackslash ref}\{\text{lab}\}, but first case creates larger active area for mouse clicking, when \text{\textbackslash 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 \text{\textbackslash label}\{(\text{label})\} before the \text{\textbackslash theorem}, \text{\textbackslash definition} etc. (macros defined with \text{\textbackslash numberedpar}) if you want to reference these numbered objects. You can't use \text{\textbackslash theorem}\{(\text{label})\} because the optional parameter is reserved to another purpose here.

You can create a reference to whatever else by commands \text{\textbackslash label}\{(\text{label})\} w\text{\textbackslash label}\{(\text{text})\}. The connection between \(\text{\textbackslash label}\) and \(\text{\textbackslash text}\) is established. The \text{\textbackslash ref}\{(\text{label})\} will print \(\text{\textbackslash text}\).

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

1.4.4 Hyperlinks, outlines

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

The last object is an external link and it is colored by \(\text{\textbackslash text}\). Other links are internal and they are colored by \(\text{\textbackslash text}\). Example:

\text{
\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 \text{\textbackslash pgborder}, \text{\textbackslash tocborder}, \text{\textbackslash citeborder}, \text{\textbackslash refborder} and \text{\textbackslash urlborder} as the triple of RGB components of the used color. Example:
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 } have to be protected by backslash in the \url argument, the other special characters ~, ^, & can be written as single character 4. You can insert the \| command in the \url argument as a potential breakpoint. If the linked text have to be different than the URL, you can use \ulink[\{\url\}]\{\text\} macro. For example: \ulink[http://petr.olsak.net/optex]{\OpTEX\ page} outputs to the text OpTEX page. 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:

\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:
  1. First sub-idea
  2. Second sub-idea
  3. Last sub-idea
- Finito

Another style can be defined by the command \sdef{\_item}{\langle style\rangle}{\langle text\rangle}. Default item can be set by \defaultitem{\langle text\rangle}. The list environments can be nested. Each new level of items is indented by next multiple of \iindent 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{\langle declaration\rangle}{\langle data\rangle} provides similar \langle declaration\rangle of tables as in \LaTeX: 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 \crl1 \tskip2pt
January & notebook & \$ 700 \cr
February & skateboard & \$ 100 \cr
July & yacht & k\$ 170 \crl
```

generates the result:
Apart from l, r, c declarators, you can use the p\{\langle 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\{\langle size\}\fL\} fit left, i.e. left justified, ragged right,
- p\{\langle size\}\fR\} fit right, i.e. right justified, ragged left,
- p\{\langle size\}\fC\} fit center, i.e. ragged left plus right,
- p\{\langle size\}\fS\} fit special, short one-line paragraph centered, long paragraph normal,
- p\{\langle size\}\fX\} fit extra, left-right justified but last line centered.

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

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

The command \cr used in the \langle 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:

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

The \tskip\{\langle dimen\}\} command works like the \noalign{\vskip \langle 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
\vvkern=1pt % space between lines in double vertical line
\hhkern=1pt % 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\{\langle number\}\} (from Plain \TeX) can help you. Another alternative is the command \mspan\{\langle number\}\}[\langle declaration\}\]\{\langle text\}\} which spans \langle number\} columns and formats the \langle text\} by the \langle declaration\}. The \langle declaration\} must include a declaration of only one column with the same syntax as common \table \langle declaration\}. If your table includes vertical rules and you want
to create continuous vertical rules by \mspan, then use rule declarators \, 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:

\frame\{\table\{|c||l||r|\} \n\mspan3\{c\}\{\bf Title\} \n\noalign{\kern\hhkern}\crli \nfirst \& second \& third \crli \nseven \& eight \& nine \crli}\}

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

\thisstable\{\tabstrut={\vrule height 20pt depth10pt width0pt} \n\baselineskip=20pt \tablinespace=0pt \rulewidth=.8pt}\}
\table\{|8\{c|\}\} \n\mspan2\{c\}\{I\} & \mspan3\{c\}\{O\} & \mspan3\{c\}\{X\} \n\mspan2\{c\}\{II\} & \mspan6\{c\}\{X\} \n\mspan2\{c\}\{III\} & \mspan4\{c\}\{O\} & \mspan4\{c\}\{X\} & \mspan4\{c\}\{X\} & \mspan2\{c\}\{X\} & \mspan2\{c\}\{O\} \n| Singular | Plural |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Neuter</td>
<td>Masculine</td>
</tr>
<tr>
<td>I</td>
<td>O</td>
</tr>
<tr>
<td>II</td>
<td>X</td>
</tr>
<tr>
<td>III</td>
<td>O</td>
</tr>
</tbody>
</table>

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

The rule width of tables and implicit width of all \vrules and \hrules can be set by the command \rulewidth=⟨dimen⟩. The default value given by \TeX is 0.4 pt.

The c, l, r and p are default “declaration letters” but you can define more such letters by \def\_tabdeclare\{⟨letter⟩\}{⟨left⟩##⟨right⟩}. More about it is in technical documentation in section 2.30.5. See the definition of the \_tabdeclarec macro, for example.

The : columns boundary declarator is described in section 2.30.1. The tables with given width can be declared by to⟨size⟩ or pxto⟨size⟩. 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\{char\}. For example \verbchar\* declares the character * for in-line verbatim markup. And you can use *\relax* for verbatim \relax (for example). Another alternative of printing in-line verbatim text is \code\{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 $\begtt...\endtt$ 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:

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

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 $\fnote$s, because these macros are specially defined in Op\TeX). 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:

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

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

$$\verbinput (12-42) \text{ program.c } % \text{ listing from program.c, only lines 12-42}$$
$$\verbinput (-60) \text{ program.c } % \text{ print from begin to the line 60}$$
$$\verbinput (61-) \text{ program.c } % \text{ from line 61 to the end}$$
$$\verbinput (-) \text{ program.c } % \text{ whole file is printed}$$
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 \hisyntax{C} activates colors for C programs. Or \verbinput \hisyntax{HTML} (-) file.html can be used for HTML or XML codes. OpTEX implements C, Lua, 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 ⟨first⟩⟨second⟩. Such comments are fully interpreted by \TeX (i.e. not verbatim). Section 2.28.1 (page 144) 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{(case-toc)}{(case-mark)}{(case-outline)}. 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}{\L Go}}}} % used in running heads
{\def\mylogo{My\L Go}} % 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 don’t have to 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 \ll 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 \iiit type ⟨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 \iiit b, \iiit i and \iiit 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. \opTeX 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 \iiis command.

Example:

```
\iiis chiquadrat $\chi$-quadrat
\iiis relax \{\texttt{\relax}\}
```

The \iiis \{\textit{equivalent}\} \{\textit{text}\} creates one entry in the “dictionary of the exceptions”. The sorting is done by the \{\textit{equivalent}\} 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{}ing

The command \texttt{\cite\[\langle \textit{label}\rangle\]} (or \texttt{\cite\[\langle \textit{label-1}\rangle,\langle \textit{label-2}\rangle,\ldots,\langle \textit{label-n}\rangle\]}) creates the citation in the form [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\[\langle \textit{labels}\rangle\]} creates the same list as \texttt{\cite\[\langle \textit{labels}\rangle\]} but without the outer brackets. Example: \texttt{\[\rcite\[\texttt{tbln}, \texttt{pg.}\] creates 4, \texttt{pg. 13}.}

The \texttt{\ecite\[\langle \textit{label}\rangle\}\{\langle \textit{text}\rangle\}} 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:

```
\def\cite[#1]{(\rcite[#1])}% \cite\[\langle \textit{label}\rangle\] creates (27)
\def\cite[#1]{^\{\rcite[#1]\} % \cite\[\langle \textit{label}\rangle\] creates^{27}
```

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\[\langle \textit{label}\rangle\]} commands.
- By \texttt{\usebib\[\langle \textit{type}\rangle\\langle \textit{style}\rangle\\langle \textit{bib-base}\rangle\]} command which reads *.bib files directly.

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

**References created manually using \texttt{\bib\[\langle \textit{label}\rangle\]} command.**

```
\bib [tst] P. Olšák. \texttt{\{it Typografický systém \TeX{}\}}
```

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\[\langle \textit{label}\rangle\] = \{\textit{mark}\}}. The spaces around equal sign are mandatory. Example:

```
\bib [tbn] = {Olšák, 2001}
```

**Direct reading of \texttt{.bib} files** is possible by \texttt{\usebib} macro. This macro reads and uses macro package librarian.tex by Paul Isambert. The usage is:
The \texttt{\usebib/c} (\texttt{\cite-order}) \texttt{\langle\textit{bib-base}\rangle} \% sorted by \texttt{\cite-order} (c=\cite), 
\texttt{\usebib/s} (\texttt{\langle\textit{style}\rangle}) \texttt{\langle\textit{bib-base}\rangle} \% sorted by \texttt{\langle\textit{style}\rangle} (s=\texttt{\langle\textit{style}\rangle}). 
\% example:
\texttt{\nocite[*]} \texttt{\usebib/s} (\texttt{\langle\textit{simple}\rangle}) op-biblist \% prints all from \texttt{op-biblist.bib}

The \texttt{\langle\textit{bib-base}\rangle} is one or more \texttt{\*\textunderscore\textit{bib}} database source files (separated by commas and without extension) and the \texttt{\langle\textit{style}\rangle} is the part of the filename \texttt{\textit{bib}-\langle\textit{style}\rangle}\texttt{.opm} where the formatting of the references list is defined. \textsc{OpTeX} supports \texttt{\langle\textit{simple}\rangle} or \texttt{\langle\textit{iso690}\rangle} styles. The features of the \texttt{\langle\textit{iso690}\rangle} style is documented in the section \texttt{2.32.6} in detail. The \texttt{\usebib} command is more documented in section \texttt{2.32.2}.

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

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

\begin{verbatim}
\bibpart {AA}
... \cite{labelX} ... \cite{labelY} ... \% They belong to AA bib-list
\usebib/c (\langle\textit{simple}\rangle) 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 (\langle\textit{simple}\rangle) my.bib \% generates BB bib-list numbered 1, 2, ...
\% \cite prints [1], [2], ... by bib-list BB
\end{verbatim}

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

1.6 Graphics

1.6.1 Colors, transparency

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

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

The color selectors work locally in groups like font selectors.

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

The basic colors \texttt{\Blue}, \texttt{\Red}, \texttt{\Cyan}, \texttt{\Yellow} etc. are defined with CMYK components using \texttt{\setcmykcolor}. On the other hand, you can define a color with three RGB components and \texttt{\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 \texttt{\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 for printing.

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\langle background\rangle\langle foreground\rangle\{\langle text\rangle\}

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}

The \transparency\langle number\rangle sets the transparency amount of following typesetting material until the current group is closed. The \langle number\rangle must be in the range 0..255, zero means no transparency (solid objects), 255 means full transparency (invisible objects). You can see the effect when overlapping one object over another.

1.6.2 Images

The \inspic {\langle filename\rangle.\langle extension\rangle} or \inspic {\langle filename\rangle.\langle extension\rangle\langle space\rangle} inserts the picture stored in the graphics file with the name \langle filename\rangle.\langle extension\rangle to the document. You can set the picture width by \picw=\langle dimen\rangle 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=\langle img/\rangle supposes that image files are in \img subdirectory. Note: the directory name must end by / in the \picdir declaration. More parameters can be included using the \picparams token list.

Inkscape\textsuperscript{6} is able to save a picture to PDF and labels of the picture to another file\textsuperscript{7}. This second file should be read by \LaTeXx to print labels in the same font as document font. \OpLaTeX supports this feature by \inkinspic {\langle filename\rangle.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

\textsuperscript{5} Printed output is more equal to the monitor preview especially if you are using ICC profile for your printer.

\textsuperscript{6} A powerful and free Wysiwyg editor for creating vector graphics.

\textsuperscript{7} Choose “Omit text in PDF and create \LaTeX file” option.
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 \[(c), (d)\]. 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.

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

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

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: second: third: text1  text2  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 \text{(transformation)} parameter includes one or more transformation commands \pdfsetmatrix, \pdfscale, \pdfrotate with their parameters. The \text{(text)} is transformed text.

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

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={\langle settings \rangle}. The default settings are:

\ovalparams={\roundness=2pt} % diameter of circles in the corners  
\fcolor=Yellow % color used for filling oval  
\lcolor=Red % line color used in the border  
\lwidth=0.5bp % line width in the border
\shadow=N % use a shadow effect
\overlapmargins=N % ignore margins by surrounding text
\hhkern=0pt \vvkern=0pt % left-right margin, top-bottom margin

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

If you need to set a parameters for the $\langle text \rangle$ (color, size, font etc.), put such setting right in front of the $\langle text \rangle$: \texttt{\smalltext settings}{\langle text \rangle}.

The $\texttt{\circ}{\langle text \rangle}$ creates a box like this $\langle text \rangle$. The $\langle ratio=1.8 \rangle$ parameter means width/height. The usage is analogical like for oval. The default parameters are

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

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

\begin{verbatim}
\clipincircle 3cm 3.5cm 6cm 7cm \{\picw=6cm \inspic{myphoto.jpg}}
\end{verbatim}

1.6.5 Putting images and texts wherever

The \texttt{\puttext} $\langle x \rangle$ $\langle y \rangle$ $\langle text \rangle$ puts the $\langle text \rangle$ shifted by $\langle x \rangle$ right and by $\langle y \rangle$ 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} $\langle x \rangle$ $\langle y \rangle$ $\langle text \rangle$ puts the text at the coordinates $\langle x \rangle$, $\langle y \rangle$. More exactly the left edge of its baseline is at that position.

The \texttt{\putpic} $\langle x \rangle$ $\langle y \rangle$ $\langle width \rangle$ $\langle height \rangle$ $\langle image-file \rangle$ puts an image given by $\langle image-file \rangle$ (including extension) of given $\langle width \rangle$ and $\langle height \rangle$ at given position (its left-bottom corner). You can write \texttt{\nospec} instead $\langle width \rangle$ or $\langle height \rangle$ if this parameter is not specified.

1.7 Others

1.7.1 Using more languages

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


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

For compatibility with $\langle language \rangle$ 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 \langle iso-code\rangle\lang commands mentioned above. Default is \enlang.

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 \langle iso-code\rangle\lang. OpTEX declares these words only for few languages: Czech, German, Spanish, French, Greek, Italian, Polish, Russian, Slovak, Hebrew 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 2.37.2.

The \makeindex command needs to know the sorting rules used in your language. OpTEX 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 \langle iso-code\ranglequotes, then the control sequences \" and \' should be used like this: \"\langle quoted text\rangle" or \'\langle quoted text\rangle' (note that the terminating character is the same but it isn’t escaped). This prints language-dependent normal or alternative quotes around \langle quoted text\rangle. The language is specified by \langle iso-code\rangle. OpTEX 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 2.37.2. 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 \langle isocode\ranglequotes\altquotes in such case.

1.7.2 Pre-defined styles

OpTEX 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 11pt and \parindent (paragraph indentation) to 1.2em. 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 \langle authors\rangle\end-line can be used when \report is declared. It prints \langle authors\rangle 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 11pt and \parindent to 0pt. 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:

\address
  \langle first line of address\rangle
  \langle second line of address\rangle
  \langle etc.\rangle
  \langle empty line\rangle

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 \langle prefixed text\rangle\address... to put \langle prefixed text\rangle before the first line of the address.
The \texttt{slides} style creates a simple presentation slides. See an example in the file \texttt{op-slides.tex}. Run \texttt{optex op-slides.tex} and see the documentation of \texttt{slides} style in the file \texttt{op-slides.pdf}.

Analogical declaration macro \texttt{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 \texttt{\input{(file-name)}} or by \texttt{\load[(file-names)]}. The first case (\texttt{\input}) is \TeX\ primitive command, it can be used in the alternative old syntax \texttt{\input filename(space)} too. The second case (\texttt{\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 \texttt{filename.opm} or \texttt{filename.tex} or \texttt{filename}, the first occurrence wins. Example:

\begin{verbatim}
\load [qrkode, scanbase]
\end{verbatim}

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

It is strongly recommended to use the \texttt{\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 \texttt{\input} primitive.

The macro packages intended to Op\TeX\ have the name *.opm. The list of packages supported by Op\TeX\ follows. Most of them are directly part of Op\TeX:\

- \texttt{math.opm} provides usable features for math typesetting and shows how to create new packages.
- \texttt{qrkode.opm} enables to create QR codes.
- \texttt{tikz.opm} does \texttt{\input tikz.tex}, i.e. loads Ti\textit{k}Z. It adds Op\TeX\-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{minim-mp.opm} enables \texttt{\directmetapost} using \texttt{minim-mp} and \texttt{minim} packages.
- \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)-\langle to\rangle]}. It prints a paragraph (or paragraphs) with neutral text. The numbers \langle number\rangle or \langle from\rangle, \langle to\rangle must be in the range 1 to 150 because there are 150 paragraphs with neutral text prepared for you. The \texttt{\lipsum} macro is equivalent to \texttt{\lorem}. Example: \texttt{\lipsum[1-150]} prints all prepared paragraphs.

If the dot follows the argument before closing \} (for example \texttt{\lipsum[3.]} or \texttt{\lipsum[3.1]}) then only first sentence from given paragraph is printed.

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{verbatim}
We are using \TeX{} because it is cool. \OpTeX{} is better than \LaTeX{}.
\end{verbatim}
1.7.6 The last page

The number of the last page (it may be different from the number of pages) is expanded by `\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”:

\footline={\hss \fixedrm \folio/\lastpage \hss}

The `\lastpage` expands to the last `\folio` which is a decimal number or Roman numeral (when `\pageno` is negative). If you need to know the total pages used in the document, use `\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 Op\TeX

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

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

1.7.8 Op\TeX tricks

The page Op\TeX tricks shows many other features of Op\TeX. They are of different nature and they are typically implemented by short chunks of macro code presented at the page.

Selected macros defined as an Op\TeX trick can be used directly from your document without copying the code chunks into your macros. It is because these macros are “registered” in Op\TeX (by `\_regtrick` internally) and if you use such a macro then Op\TeX automatically loads the appropriate code chunk from an external file. These macros are listed here. More information about them are accesible via the external links.

- `\algol` enables to create pseudocode listings.
- `\beglua, \begLUA, \logginglua` writing LUA codes as LUA codes.
- `\cancel` prints a given text and the line/cross line over the text.
- `\createfile, \begfile, \endfile` writes a code from the document to the given file.
- `\colortab` colored cells in the table.
- `\correctvsize` sets `\vsize` to fit lines exactly to pages.
- `\crtop, \crmid, \crbot` specific design of tables: only horizontal rules with different thickness.
- `\crx` alternating colored lines in tables.
- `\directoutput` puts boxes to standalone pages adapted to the box dimesions.
- `\easylist` the depth of list is given by the number of `*`.
- `\fread, \fullcite` citations by full bibliographic records.
- `\framedblocks` redefines `\begblock, \endblock` to create blocks in frames splittable to pages.
- `\ignoreinspic` the `\inspic` commands stop loading images, they are replaced by gray frames.
- `\import` allows to have subsets of document input files in separate directories.
- `\ispageodd` tests, if the current point is at odd page regardless of asynchronous processing.
- `\incrpp, \themp, \thepplast, \truepage` does per-page counting of objects.
- `\keystoke` prints given text in a keystroke-like frame.
- `\longtable` alows to break a table to more pages and repeates header.
- `\makeLOF, \makeLOT, \captionF, \captionT` create list of tables and list of figures similar to `\maketoc`.
- `\pstart, \pend` display line numbers of the marked text in the margin.
- `\settabs, \tabs` macros emulate tabulators of old typewriters.
- `\showpglists` shows good organized list of nodes of given pages to the log file.
- `\truepage` positions of table items are nodes, they can be used for drawing.
\texttt{\textbackslash tnote} creates notes for table data printed just after the table.
\texttt{\textbackslash tlineref} verbatim lines referenced in text.
\texttt{\textbackslash vcent}, \texttt{\textbackslash vbot} prints paragraphs in tables vertically centered or placed at bottom.
\texttt{\textbackslash twoblocks} allows printing bilingual texts in two columns vertically aligned.

1.8 Summary

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

\texttt{\textbackslash maketoc} % table of contents generation
\texttt{\textbackslash ii item1,item2} % insertion the items to the index
\texttt{\textbackslash makeindex} % the index is generated

\texttt{\textbackslash label [labname]} % link target location
\texttt{\textbackslash ref [labname]} % link to the chapter, section, subsection, equation
\texttt{\textbackslash pgref [labname]} % link to the page of the chapter, section, ...

\texttt{\textbackslash caption/t} % a numbered table caption
\texttt{\textbackslash caption/f} % a numbered caption for the picture
\texttt{\textbackslash eqmark} % a numbered equation

\texttt{\textbackslash begitems} % start a list of the items
\texttt{\textbackslash enditems} % end of list of the items
\texttt{\textbackslash begblock} % start a block of text
\texttt{\textbackslash endblock} % end of block of text
\texttt{\textbackslash begtt} % start a verbatim text
\texttt{\textbackslash endtt} % end verbatim text
\texttt{\textbackslash verbchar X} % initialization character X for in-text verbatim
\texttt{\textbackslash code} % another alternative for in-text verbatim
\texttt{\textbackslash verbatiminput} % verbatim extract from the external file
\texttt{\textbackslash begmulti num} % start multicolumn text (num columns)
\texttt{\textbackslash endmulti} % end multicolumn text

\texttt{\textbackslash cite [labnames]} % refers to the item in the lits of references
\texttt{\textbackslash rcite [labnames]} % similar to \texttt{\textbackslash cite} but [ ] are not printed.
\texttt{\textbackslash sortcitations | \textbackslash shortcitations | \textbackslash nonumcitations} % cite format
\texttt{\textbackslash bib [labname]} % an item in the list of references
\texttt{\textbackslash usebib/\?} % (style) bib-base % direct using of \texttt{.bib} file, ? in \{s,c\}

\texttt{\textbackslash load [filenames]} % loading macro files
\texttt{\textbackslash fontfam [FamilyName]} % selection of font family
\texttt{\textbackslash typosize [font-size/baselineskip]} % size setting of typesetting
\texttt{\textbackslash typoscale [factor-font/factor-baselineskip]} % size scaling
\texttt{\textbackslash thefontsize [size] \textbackslash thefontsizecale [factor]} % current font size

\texttt{\textbackslash inspic file.ext} % insert a picture, extensions: \texttt{.jpg}, \texttt{.png}, \texttt{.pdf}
\texttt{\textbackslash table \{rule\}\{data\}} % macro for the tables like in \texttt{LaTeX}

\texttt{\textbackslash fnote \{text\}} % footnote (local numbering on each page)
\texttt{\textbackslash amnote \{text\}} % note in the margin \texttt{(left or right by page number)}

\texttt{\textbackslash hyperlinks \{color-in\}\{color-out\}} % PDF links activate as clickable
\texttt{\textbackslash outlines \{level\}} % PDF will have a table of contents in the left tab

\texttt{\textbackslash magscale[\textbackslash factor]} % resize typesetting, line/page breaking unchanged
\texttt{\textbackslash margins[\textbackslash pg format (left, right, top, bottom)]unit} % margins setting
\texttt{\textbackslash report \textbackslash letter \textbackslash slides} % style declaration macros
1.9 API for macro writers

All \TeX{} primitives and almost all \OFTeX{} macros are accessible by two names: \texttt{\_foo} (public or user namespace) 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 2.2.1.

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. The \texttt{\_codedecl} equivalent exists too, but it is not explicitly mentioned here.

If you are writing a document or macros specific for the document, then use simply public name space (\texttt{\_foo}). If you are writing more general macros, then you should declare your own name space by \texttt{\_namespace} macro and you have to follow the naming discipline described in sections 2.2.1 and 2.2.3.

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.

\texttt{\addto} \texttt{\_macro}{{\langle text\rangle}} adds \langle text\rangle at the end of \texttt{\_macro} body, \texttt{\heado \_macro}{{\langle text\rangle}} puts \langle text\rangle at the begin.
\texttt{\def} \texttt{\char}{{\langle body\rangle}} defines \texttt{\char} active character with meaning \langle body\rangle.
\texttt{\afterfi} \texttt{\{text\}} ignored \texttt{\fi} expands to \texttt{\fi}{{\langle text\rangle}}.
\texttt{\basename} \texttt{\_curri} returns the name of the file currently read.
\texttt{\bp \{\texttt{\dimen expression}\}} expands \TeX{} dimension to decimal number in \texttt{\bp} without unit.
\texttt{\casesof} \texttt{\{info\}} is used at beginning of macro files.
\texttt{\colordef} \texttt{\_macro} {{\langle \{mix\ of\ colors\}\rangle}} declares \texttt{\_macro} as color switch.
\texttt{\cs} {\langle \{string\}\rangle} expands \langle \{string\}\rangle.
\texttt{\cstoch} {\langle \{sequence\}\rangle} converts \langle \{sequence\}\rangle to \langle \{character\}\rangle if there was \texttt{\let}{{\langle sequence\rangle}=\langle character\rangle}.
\texttt{\doc \ldots \_doc} encloses documentation text in the macro code.
\texttt{\eoldef} \texttt{\_macro} \{\langle body\rangle\} defines \texttt{\_macro} with parameter separated to end of line.
\texttt{\endcode} closes the part of macro code in macro files.
\texttt{\_endnamespace} closes name space declared by \texttt{\_namespace}.
\texttt{\eqbox} \{\langle label\rangle\}{{\langle text\rangle}} creates \texttt{\bbox}{{\langle text\rangle}} with common width across whole document.
\texttt{\exp} \{\{expression\}\} expands to result of the \langle expression\rangle with decimal numbers.
\texttt{\fontdef} \langle \{font\ spec\}\rangle declares \texttt{\font} as font switch.
\texttt{\fontlet \fa=\fb} \langle \{sizespec\}\rangle declares \texttt{\fa} as the same font switch like \texttt{\fb} at given \langle \{sizespec\}\rangle.
\texttt{\foreach} \{\langle list\rangle\} \texttt{\do} \{\langle parameters\rangle\}{\langle what\rangle\}} is expandable loop over \langle list\rangle.
\texttt{\foreachdef} \texttt{\_macro} \{\langle parameters\rangle\}{\langle what\rangle\}} declares expandable \texttt{\_macro} as loop over \langle list\rangle.
\texttt{\form} \{\langle from\rangle\} \texttt{\to} \{\langle to\rangle\} \texttt{\do} \{\langle what\rangle\} is expandable loop with numeric variable.
\texttt{\incr} \{\texttt{\counter}\} increases and \texttt{\decr} \{\texttt{\counter}\} decreases \langle \texttt{\counter}\rangle by one globally.
\texttt{\ignoreone} \texttt{\{\texttt{\ignoretwo}\}} \texttt{\{\texttt{\ignorethree}\}} \texttt{\{\texttt{\ignorefour}\}} ignores given parameter.
\texttt{\expandafter} \texttt{\ignorept} \texttt{\the\{\dimen\}} expands to decimal number \{\dimen\} without pt.
\texttt{\isempty} \texttt{\{\texttt{\isempty\}} \texttt{\isequal} \texttt{\{\texttt{\ismacro} \texttt{\isdefined} \texttt{\isinlist} \texttt{\isfile} \texttt{\isfont} do various tests. Example: \texttt{\isaninlist\{\langle text\}\}} \texttt{\isfalse done with \texttt{\iftrue} does \texttt{\iftrue} if \langle text\} is in \langle list\}. \texttt{\isnanextchar} \{\langle char\}\}{{\langle text1\rangle}} \{\langle text2\rangle\}} performs \langle text1\} if next character is \langle char\}, else \langle text2\}. \texttt{\kv} \{\{key\}\} expands to value when key-value parameters are used. See also \texttt{\iskv}, \texttt{\readkv}, \texttt{\kvx}, \texttt{\nokvx}.
\langle loop \ldots \} \langle \texttt{\repeat} is classical Plain \TeX{} loop.
\texttt{\mathtyes} \{\langle \texttt{\math list}\rangle\} enables to create macros dependent on current math style.
\texttt{\_namespace} \{\langle pkg\rangle\} declares name space used by package writers.
\texttt{\newcount}, \texttt{\newdimen} etc. are classical Plain \TeX{} allocators.
\texttt{\newif} \texttt{\iffoo\} declares boolean \texttt{\iffoo} as in Plain \TeX{}.
\texttt{\newif\_\iffoo\} declares boolean \_\iffoo.
\texttt{\nospaceafter\_macro}, \texttt{\nospacefuturelet}: they ignore the following optional space.
\texttt{\opinput} \{\langle filename\}\} reads file like \texttt{\input} but with standard catcodes.
\texttt{\optdef} \texttt{\_macro} \{\texttt{\{opt-default\}}\} \{\langle parameters\rangle\}{\langle body\rangle\}} defines \texttt{\_macro} with \texttt{\{opt-parameter\}}.
\texttt{\opwarning} \{\langle text\}\} prints \langle text\} to the terminal and \texttt{.log} file as warning.
\texttt{\poss} \{\langle label\}\} \texttt{\posy} \{\langle label\}\} \texttt{\posx} \{\langle label\}\} provide coordinates of absolute position of the \texttt{\setpos} \{\langle label\}\}. \texttt{\private} \{\{sequence\}\} \ldots \} \{\texttt{\} declares \{\{sequence\}\} for private name space.
\texttt{\public} \{\{sequence\}\} \ldots \} \{\texttt{\} declares \{\{sequence\}\} for public name space.
\texttt{\replstring} \{\langle macro\rangle\}\{\langle stringA\rangle\}\{\langle stringB\rangle\} replaces all \{\langle stringA\}\} to \{\langle stringB\} in \texttt{\_macro}.
\texttt{\def} \{\{string\}\} \{\langle parameters\rangle\}{\langle body\rangle\}} replaces \langle \{string\}\} with \langle \{parameters\}\}. \texttt{\setetable} and \texttt{\restoreetable} manipulate with stack of catcode tables.
\let \{stringA\}\{stringB\} behaves like \let\{stringA\}=\{stringB\}
\xdef \{string\} \{parameters\} \{body\} behaves like \xdef\{string\}\{parameters\}\{body\}.
\trycs \{string\}\{text\} expands \{string\} if it is defined else expands \{text\}.
\uset \{one\}, \use \{second\}(\{one\})(\{two\}) uses given parameter.
\xdef \{parameters\} \{body\} behaves like \xdef\{parameters\}\{body\}.
\trycs \{text\} expands \{text\} if it is defined else expands \{text\}.
\wlog \{text\} writes \{text\} to .log file.
\wterm \{text\} writes \{text\} to the terminal and .log file.
\xargs \{what\} \{token\} \{token\} ... ; 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 \f@@t are removed and mostly replaced by control sequences prefixed by _ (like \_this). Only a basic set of old Plain \TeX control sequences like \p@, \z@, \dimen@ are provided but not recommended for new macros.

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 user 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 \"', \'=, \-=, \^, \~, \` 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.

1.11 Related documents
• Typesetting math with Op\TeX – More details about math typesetting.
• \TeX in a Nutshell – Summary about \TeX principles, \TeX primitive commands etc.
• Op\TeX catalog – All fonts collected to \fontfam families are shown here.
• OMLS – Op\TeX Markup Language Standard.
• Op\TeX - tips, tricks, howto – Tips of macro codes for various purposes.

\footnote{8 The math accents macros like \acute, \bar, \dot, \hat still work.}
\footnote{9 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

```
lutex -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

```
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.

```
%% 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 `\^=7 %
\catcode `\^^K=7 % circumflex and uparrow are for superscripts
\catcode `\^^A=8 % downarrow is for subscripts
\catcode `\``=10 % ASCII tab is a blank space
\catcode `\_=11 % underline can be used in control sequences
\catcode `\%=13 % tilde is active
\catcode `\~a=13 % non-breaking space in Unicode
\catcode `\~c=12 % normal character
\def\optexversion{1.13 Nov 2023}
\def\fmtname{OpTeX}
\let\fmtversion=\optexversion
```

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 \optexversion and \fmtname are defined.

```
% \texttt{\textbackslash optexversion\textbar \texttt{\textbackslash fmtname}}\texttt{\textbar OPTEX: main initialization file}
\def\optexversion{1.13 Nov 2023}
\def\fmtname{OpTeX}
```

We check if Lua\TeX engine is used at -ini state. And the `^^J character is set as \newlinechar.
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 \texttt{\_everyjob} register is initialized and the format is saved by the \texttt{\_dump} command.

2.2 Basic principles of OpT\!\TeX\! sources

2.2.1 Concept of namespaces of control sequences

OpT\!\TeX\! sets the category code of the “\_” character to 11 (letter) and it is never changed.\footnote{This is only singular exception form category codes given by plain \TeX.} So, we can always construct multiletter control sequence names from letters \( A \)–\( Z \), \( a \)–\( z \), and \( \_ \). The “\_letter \_” works in math mode as a subscript constructor because it is set as math active character (see section 2.15).

We distinguish following namespaces for multiletter control sequences:

- Only alphabetical names are in the \textit{public namespace}. They are intended for end users when creating a document. Sometimes it is called \textit{user namespace} too. For example \texttt{\_hbox}, \texttt{\_fontfam}, \texttt{\_MyMacro}.
- Only alphabetical lowercase names prefixed by single “\_” are in the \textit{private namespace}. It is used in OpT\!\TeX\! internal macros. For example \texttt{\_hbox}, \texttt{\_fontsel}.
- Names in the form \texttt{\_\langle pkg\_\rangle\_\langle name\_\rangle} are in the \textit{package namespace}, see section 2.2.3. For example \texttt{\_qr\_size}, \texttt{\_math\_alist}.
- Names starting with two “\_” are in the \textit{reserved namespace}. They can be used for internal control sequences in font family files or in similar cases.
- Other names which include “\_” but not as the first character can be used too, but with care, see the end of this section.

All \TeX\! primitives are initialized with two control sequences with the same meaning: \textit{prefixed} control sequence (in private namespace, for example \texttt{\_hbox}) and \textit{unprefixed} control sequence (in public namespace, for example \texttt{\_hbox}). All OpT\!\TeX\! macros intended for end users are initialized in these two forms too, for example \texttt{\_ref} and \texttt{\_ref}.

Users can declare any control sequences in the public namespace without worrying that OpT\!\TeX\! behavior is changed. This is because OpT\!\TeX\! uses exclusively prefixed control sequences in its macros. For example, a user can declare \texttt{\_def\_\_f\_\langle finito\_\rangle} and nothing bad happens, if the user doesn’t use \texttt{\_f\_i} in its original primitive meaning. You don’t have to know all \TeX\! primitives and OpT\!\TeX\! macros, you can declare control sequences for your use in the public namespace without limitations and nothing bad will happen.

You can use control sequences from private or package namespace in a “read-only manner” without changing OpT\!\TeX\! behavior too. On the other hand, if you re-define a control sequence in the private name space, the OpT\!\TeX\! behavior can be changed. You can do it but we suppose that you know what you are doing and what OpT\!\TeX\! behavior is changed.

All multiletter control sequences declared by OpT\!\TeX\! are defined in the private namespace first (\texttt{\_\_def\_\_macro\{...\}}). If the declared control sequences are intended for end users too then they are exported to the public namespace after that. It is done by the \texttt{\_public} macro:

\begin{verbatim}
\_public \langle list of control sequences\rangle ;
\end{verbatim}

For example \texttt{\_public \_foo \_bar ;} does \texttt{\_let\_foo=\_foo, \_let\_bar=\_bar.}

There is an exception of the above mentioned principle. Control sequences which are alternatives to math characters (\texttt{\_alpha}, \texttt{\_forall}, \texttt{\_subset} etc.) are declared only in public name space if they are not used in any internal OpT\!\TeX\! macros.
The macro \priv makes the reverse job of \pub with the same syntax. For example \priv \foo \bar ; does \let _foo=\foo, \let _bar=\bar. This should be used when an unpre-
fixed 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 \pub (or \priv) macro is used.

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).

It is very tempting to use control sequence names with _ in order to distinguish more words in the
sequence name. If the first character isn’t _ then such a name is outside private and package namespaces,
so they can be used for various purposes. For example \my_control_sequence. But there is an exception:
control sequences in the form \langle word \rangle or \langle word \rangle\langle one-letter \rangle, where \langle word \rangle is a sequence of letters, are inaccessible, because they are interpreted as \langle word \rangle followed by _ or as \langle word \rangle followed by \langle one-letter \rangle.
This feature is activated because we want to write math formulae as in plain \TeX, for example:

\int_a^b \ldots is interpreted as \int _a^b
\max_M \ldots is interpreted as \max _M
\alpha_{ij} \ldots is interpreted as \alpha _{ij}

It is implemented using Lua code at input processor level, see the section 2.15 for more details. You can
deactivate this feature by \mathsbon. 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. You can activate this feature
again by \mathsboff. The effect will take shape from next line read from input file.

2.2.2 Macro files syntax

Segments of Op\TeX macros or external macro packages are stored in files 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 \langle short title \rangle \langle version \rangle

If the mentioned \sequence is undefined then \ codedecl prints the message
0:\langle file name \rangle \langle short title \rangle \langle version \rangle

to the log file and \TeX continues with reading the following macros. If the \sequence is defined, then
\ codedecl acts like \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 definitions. The documentation text
should be here. It is ignored when macros are read.

The \doc ... \cod parts can be printed after \loaddoc using \printdoc macro, see section 2.40.
If you have created a documented macro file pkgname.opm then you can put macros for creating your document-
between first pair of \doc ... \cod used after \endcode. These macros should \loaddoc and
must be finished by \bye. Then you have code+documentation together in a single file and user can
generate the documentation of your package by \doct gen used at command line:

\optex -jobname pkgname-doc 'docgen pkgname'

Example of a \doc ... \cod code used for creating the documentation using \doct gen can be found in the
math.opm file. You can see its documentation, especially section about creating packages.

2.2.3 Name spaces for package writers

Package writer should use internal names in the form \langle pkg \rangle \langle sequence \rangle, where \langle pkg \rangle 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 \langle pkg \rangle is declared at the beginning of the macro file then all occurrences of \foo will be replaced by \langle pkg \rangle \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.

\_nspublic \{list of sequences\} ; does \let\foo = \_\{pkg\}_foo for each given sequence 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 \def\macro{...} and then set it to the public namespace by \_nspublic \macro;.

It could happen that a package writer needs to declare a control sequence (say \foo) directly without setting it in \_\{pkg\}_foo namespace followed by using \_nspublic. The \_newpublic prefix should be used in this case, for example \_newpublic\_def\foo or \_newpublic\_chardef\foo. The \_newpublic\{do\}\sequence prints a warning if the declared \sequence is defined already and then runs \{do\}\sequence. The reason of the warning is the same as when \_nspublic warns about doing re-declaration of control sequences already declared.

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).

### 2.2.4 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:

- Don’t use control sequences from the public namespace in the macro bodies if there is no explicit and documented reason to do this.
- Don’t declare control sequences in the public 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 2.2.3.
- 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 \_nspublic \_nsprivate \macro;.

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 math.opm is a good example of how an external macro file for OpTeX can look like. Another good and short example is here.

### 2.2.5 The implementation of the namespaces and macros for macro-files

All \TeX primitives have alternative control sequence \_hbox \_string, ...
\texttt{\_ea} is useful shortcut for \texttt{\expandafter}. We recommend to use always the private form of \texttt{\_ea} because there is high probability that \texttt{\_ea} will be redefined by the user.

We define the macros \texttt{\_namespace \{\textit{pkg label}\}}, \texttt{\resetnamespace \{\textit{pkg label}\}}, \texttt{\_endnamespace}, \texttt{\_pkglabel}, \texttt{\_nspublic}, and \texttt{\_nsprivate} for package writers, see section 2.2.3.
Each macro file should begin with _codedecl \macro {⟨info⟩}. If the \macro is defined already then the \endinput protects to read such file more than once. Else the ⟨info⟩ is printed to the terminal and the file is read. The _encode is defined as \endinput in the optex.ini file. \wterm {⟨text⟩} prints the ⟨text⟩ to the terminal and to the .log file, \wlog {⟨text⟩} prints the ⟨text⟩ only to the .log file (as in plain TeX).

\currfile returns the name of the current input file including its path.
\basefilename\currfile returns base name of the current file, without its path and extension.
\nofilepath ⟨text⟩/⟨with⟩/⟨slashes⟩/\_fin expands to the last segment separated by slashes.
\nofileext ⟨filename⟩.\_fin expands to the file name without extension.

We define \_fin as a useless macro. Suppose that its meaning will be never used for another control sequence. You can use \_fin as a final delimiter of a list of tokens and your macro can ask \ifx\_fin#1 in order to decide that the list of tokens is finalized.

2.3 pdfTeX initialization

Common pdfTeX primitives equivalents are declared here. Initial values are set.
\_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

\_protected\_def \_pdftexversion {\_numexpr 140 \_relax}
\_protected\_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}
\_protected\_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 \_pdfreobj {\_pdffeedback reobj}
\_protected\_def \_pdffannot {\_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 startthread}
\protected\_def \_pdfendthread {\_pdffeedback endthread \_relax}
\protected\_def \_pdfinfo {\_pdffeedback info}
\protected\_def \_pdfcatalog {\_pdffeedback catalog}
\protected\_def \_pdfnames {\_pdffeedback names}
\protected\_def \_pdfincludechars {\_pdffeedback includechars}
\protected\_def \_pdffontattr {\_pdffeedback fontattr}
\protected\_def \_pdfmapfile {\_pdffeedback mapfile}
\protected\_def \_pdfmapline {\_pdffeedback mapline}
\protected\_def \_pdftrailer {\_pdffeedback trailer}
\protected\_def \_pdfglyphtounicode {\_pdffeedback glyphtounicode}
\_protected\_edef \_pdfcompresslevel {\_pdffeedback compresslevel}
\_protected\_edef \_pdfobjcompresslevel {\_pdffeedback objcompresslevel}
\_protected\_edef \_pdfdecimaldigits {\_pdffeedback decimaldigits}
\_protected\_edef \_pdfgamma {\_pdffeedback gamma}
\_protected\_edef \_pdfimageresolution {\_pdffeedback imageresolution}

\_directlua {tex.enableprimitives('pdf', {"tracingfonts"})}
2.4 Basic macros

We define first bundle of basic macros.

2.4 Basic macros.

We define first bundle of basic macros.
The \_public \_loggingall \_tracingall ;
\_def \_loggingall \{\_tracingcommands=3 \_tracingstats=2 \_tracingpages=1 \_tracingoutput=1 \_tracingscans=3 \_tracinglostchars=2 is already set \_tracingrestores=1 \_tracingscans=3 \_tracingparagraphs=1 \_tracingassigns=1 \_tracingifs=1 \_tracinggroups=1 \_tracingassigns=1 \_tracingonline=1 \_loggingall\}
\_public \_loggingall \_tracingall ;

The \_optexversion and \_fmtname are defined in the optex.ini file. Maybe, somebody will need a private version of these macros. We add \_banner used in \_everypage and in \_docgen

\_public \_decr \_incr ;
\_def \_decr #1{\_global\_advance#1by-1 }
\_def \_incr #1{\_global\_advance#1by1 }

\_public \_addto \_aheadto ;
\_long\_def \_addto #1#2{\_edef#1{\_unexpanded{#2}\_unexpanded\_ea{#1}}}
\_long\_def \_aheadto #1#2{\_ea\_def\_ea#1\_ea{#1#2}}
\_long\_def \_cs #1{\_cs\_name#1\_endcsname}
\_long\_def \_trycs #1#2{\_ifcsname #1\_endcsname \_csname #1\_ea\_endcsname \_else \_undefined \_fi}
\_public \_cs \_trycs ;
\_def \_cs #1{\_cs\_name#1\_endcsname}
\_def \_trycs #1#2{\_ifcsname #1\_ea\_endcsname \_begincsname#2\_endcsname \_else \_undefined \_fi}
\_public \_cs \_trycs ;

\_def \_addto \_macro\{\_text\} adds \_text to your \_macro, which must be defined.
\_def \_aheadto \_macro\{\_text\} defines \_macro as \_text followed by the original \_macro body.
\_def \_incr \_counter increases \_counter by one globally. \_decr \_counter decreases \_counter by one globally.

\_def \_opwarning #1{\_wterm{WARNING l.\_the\_inputlineno: #1.}}
\_public \_opwarning ;

\_def \_loggingall\{\_tracingcommands=3 \_tracingstats=2 \_tracingpages=1 \_tracingoutput=1 \_tracingmacros=3 \_tracinglostchars=2 is already set \_tracingrestores=1 \_tracinglostcharstokens=1 \_tracingifs=1 \_tracinggroups=1 \_tracingassigns=1 \_tracingonline=1 \_loggingall\}
\_public \_loggingall \_tracingall ;
\_def\_byehook{\_ifx\_initunifonts\_relax \_relax\_else \_opwarning{Unicode font was not loaded}\_fi
\_immediate\_closeout\_reffile
\_edef\_tmp{\_mdfive{\_jobname.ref}}%
\_ifx\_tmp\_prevrefhash\_else \_opwarning{Try to rerun,\_jobname.ref file was \_ifx\_prevrefhash\_empty created\_else changed\_fi}\_fi
}

2.5 Allocators for T\(E\)X registers

Like plainT\(E\)X, the allocators \texttt{\newcount}, \texttt{\newwrite}, etc. are defined. The registers are allocated from 256 to the \texttt{\_mai\langle type\rangle} which is 65535 in LuaT\(E\)X.

Unlike in PlainT\(E\)X, the mentioned allocators are not \texttt{\outer}. User can use \texttt{\dimen0} to \texttt{\dimen200} and similarly for \texttt{\skip}, \texttt{\muskip}, \texttt{\box}, and \texttt{\toks} directly. User can use \texttt{\count20} to \texttt{\count200} directly too. This is the same philosophy as in old plainT\(E\)X, but the range of directly used registers is wider.

Inserts are allocated from 254 to 201 using \texttt{\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.

\texttt{\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.

Each allocation macro needs its own counter.
The common allocation macro `\_allocator ⟨sequence⟩ {⟨type⟩} ⟨primitive declarator⟩` 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.

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

The `\newinsert` macro is defined differently than others.

Other allocation macros `\newmarks`, `\newattribute` and `\newcatcodetable` have their counter allocated by the `\newcount` macro. `\_noattr` is constant "$\text{FFFFF}$", i.e. unused attribute.
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 \zo and \zoskip (equivalents to \z and \zoskip) are declared here and used in some internal macros of OpTeX for improving speed.

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 \xxxtrue or \xxxfalse 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 \_xxxtrue and \_xxxfalse. This means that it is usable for the internal namespace (_prefixed macros).

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

\if⟨something⟩ \afterfi{⟨result is true⟩} \else \afterfi{⟨result is false⟩} \fi

Nested \if, \afterfi{⟨if..⟩}\afterfi{⟨if..⟩}\fi are possible. Another approach is mentioned in OpTeX trick 0098 which also solves the \fi in \if problem.

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.

If-macros.opm
60 \_long\_def \_loop #1 \_repeat{\_def \_body{#1}\_iterate}
61 \_long\_def \loop #1\repeat{\_def \_body{#1}\_iterate}
62 \_let \_repeat\_fi % this makes \loop...\if...\repeat skippable
63 \_let \repeat\_fi
64 \_def \_iterate {\_body \_ea \_iterate \_fi}

\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 parameters are declared by \langle parameter-mask \rangle. Examples:

\foreach \(a,1\)(\(b,2\))(\(c,3\))\do (\#1,\#2){\#1=}\#2}
\foreach \textword1,\textword2,\textword3,\do \#1,{\textword1 is set as \textword2}.
\foreach \textA=\textword1 \textB=\textword2 \do \#1=\#2 {"\#1 is set as \#2"}.

\foreach \langle list \rangle \do \langle what \rangle is equivalent to \foreach \langle list \rangle \do \#1\{\langle what \rangle \}.

Recommendation: it is better to use private variants of \foreach. When the user writes \input tikz then \foreach macro is redefined in each TikZ environment. The private variants use \_do separator instead \do separator.

If-macros.opm
89 \newcount \frnum % the numeric variable used in \fornum
90 \_def \_fornum#1..#2\_do{\_fornumstep 1:#1..#2\_do}
91 \_long\_def \fornumstep#1:#2..#3\do#4{\_putforstack
92 \_immediateassignment \_long\_gdef \_fbody##1{\_testparam##1..\_iftrue \_do \#3\_ea \_fbody\_fi%
93 \_do \#1\_2\_finbody \_getforstack
94 }
95 \_def \_testparam#1#2#3\_iftrue{\_ifx###1\_empty\_ea \_finbody\_else}
96 \_def \_fbody#1#2\_finbody{%
97 }
98 \_long\_def \fornum#1..#2\_do\{\_isempty{\#2}\_iftrue
99 \_afterfi{\_foreachA{\#1}{\_the\_numexpr\#1\_2\_fra}\_else \_afterfi{\_foreachA{\#1}{\_do \#2}}\_fi}
100 }
101 \_long\_def \fornum#1..#2\_do2\{\_isempty{\#2}\_iftrue
102 \_afterfi{\_foreachA{\#1}{\_the\_numexpr\#1\_2\_fr}\_else \_afterfi{\_foreachA{\#1}{\_do \#2 \_do}}\_fi}

\fornum \langle from \rangle..\langle to \rangle \do \langle what \rangle or \fornumstep \langle num \rangle: \langle from \rangle..\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.

If-macros.opm

The \foreach and \fornum macros can be nested and arbitrary combined. When they are nested then use \#1 for the variable of nested level, \#\#1 for the variable of second nested level etc. Example:

\foreach \textABC \do \{\fornum 1..5 \do \{\textletter: \#1, \textnumber: \#1. \} \}

44
Implementation note: we cannot use \TeX-groups for nesting levels because we want to do the macros expandable. We must implement a special for-stack which saves the data needed by \foreach and \fornum. The _putforstack is used when \foreach is initialized and \getforstack is used when the \foreach 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⟩} {⟨what⟩} which can be used by \macro {⟨list⟩}. The macro reads repeatedly parameters from ⟨list⟩ using ⟨parameter-mask⟩ and does ⟨what⟩ 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(⟨params⟩) \iftrue ⟨codeA⟩ \else ⟨codeB⟩ \fi
or
\issomething(⟨params⟩) \iffalse ⟨codeB⟩ \else ⟨codeA⟩ \fi

The \else part is optional. The ⟨codeA⟩ is processed if \issomething(⟨params⟩) generates true condition. The ⟨codeB⟩ is processed if \issomething(⟨params⟩) 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⟨⟨text⟩⟩ \iftrue is true if the ⟨text⟩ is empty. This macro is expandable.
\istoksempty⟨⟨tokens variable⟩⟩ \iftrue is true if the ⟨tokens variable⟩ is empty. It is expandable.
\isequal\{(textA)\}\{\{textB\}\} \texttt{\iftrue} is true if the \texttt{\{textA\}} and \texttt{\{textB\}} are equal, only from strings point of view, category codes are ignored. The macro is expandable.

\ismacro \macro{\text}\\texttt{\iftrue} is true if macro is defined as \texttt{\{text\}}. Category codes are ignored in this testing. The macro is expandable.

\isdefined\{(\csname)\}\texttt{\iftrue} is true if \texttt{\{\csname\}} is defined. The macro is expandable.

\isinlist\list{\{\text\}}\texttt{\iftrue} is true if the \texttt{\{\text\}} is included the macro body of the \list. The category codes are relevant here. The macro is expandable.

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

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

The macro \isnextchar \{\char\}\{\{codeA\}\}\{\{codeB\}\} has a different syntax than all other is-macros. It executes \{\texttt{\{codeA\}}\} if next character is equal to \texttt{\{char\}}. Else the \texttt{\{codeB\}} is executed. The macro is expandable.

\casesof \{\token\}\{\list of cases\} implements something similar to the \texttt{\switch} command known from C language. It is expandable macro. The \texttt{\list of cases} is a list of arbitrary number of pairs in the format
\{token\} \{(what to do)\} which must be finalized by the pair \finc \{(what to do else)\}. The optional spaces after \{token\}s and between listed cases are ignored. The usage of \casesof looks like:

\casesof \{token\}
  \{token-1\} \{(what to do if token=token-1)\}
  \{token-2\} \{(what to do if token=token-2)\}
  ...
  \{token-n\} \{(what to do if token=token-n)\}
  \finc \{(what to do in other cases)\}

The meaning of tokens are compared by if primitive. The parts \{what to do\} can be finalized by a macro which can read more data from the input stream as its parameters.

\casesof \{string\} \{list of cases\} behaves like \casesof but it compares phrases with the given \{string\} using \isequal. The \{list of cases\} includes pairs \{\{phrase\} \{what to do if string=phrase\}\} finalized by a pair \finc \{what to do else\}.

\xcasesof \{list of pairs\} extends the features of the macro \casesof. Each pair from the \{list of pairs\} is in the format \{(if statement) \{what to do\}\}, only the last pair must have the different format: \finc \{what to do else\}. The \{if statement\} can be arbitrary primitive \if\* condition (optionally prefixed by \unless) and it must be closed in its expansion. It means that \{ifnum\mycount>0\} is bad, \{ifnum\mycount>0 \} is correct. Optional spaces between parameters are ignored. Example:

\message {The \tmpnum has \xcasesof
  \{ifnum\tmpnum>0 \} \{positive\}
  \{ifnum\tmpnum=0 \} \{zero\}
  \finc \{negative\} value}

The \xcasesof macro works with principle: first true condition wins, next conditions are not evaluated.

\xcasesof \{list of pairs\} \{list of pairs\} extends the features of the macro \casesof. Each pair from the \{list of pairs\} is in the format \{(if statement) \{what to do\}\}, only the last pair must have the different format: \finc \{what to do else\}. The \{if statement\} can be arbitrary primitive \if\* condition (optionally prefixed by \unless) and it must be closed in its expansion. It means that \{ifnum\mycount>0\} is bad, \{ifnum\mycount>0 \} is correct. Optional spaces between parameters are ignored. Example:

\message {The \tmpnum has \xcasesof
  \{ifnum\tmpnum>0 \} \{positive\}
  \{ifnum\tmpnum=0 \} \{zero\}
  \finc \{negative\} value}

The \xcasesof macro works with principle: first true condition wins, next conditions are not evaluated.

2.7 Setting parameters

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

- primitive registers used in build-in algorithms of \TeX{},
- registers declared and used by \TeX{} 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. \TeX{} declares two control sequences for each register: prefixed (private) and unprefixed (public). \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 \TeX{} macros still work without change.
### 2.7.1 Primitive registers

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_parindent=20pt</td>
<td>% indentation of paragraphs</td>
<td></td>
</tr>
<tr>
<td>_pretolerance=100</td>
<td>% parameters used in paragraph breaking algorithm</td>
<td></td>
</tr>
<tr>
<td>_tolerance=200</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_badness=1000</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_doublehyphenpenalty=10000</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_finalhyphenpenalty=5000</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_adjdemerits=10000</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_uchyph=1</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_defaulthyphenchar=<code>\-</code></td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_defaultskewchar=-1</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_hfuzz=0.1pt</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_vfuzz=0.1pt</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_overfullrule=5pt</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_linepenalty=10</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_hyphenpenalty=50</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_exhyphenpenalty=50</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_binoppenalty=700</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_relpenalty=500</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_brokenpenalty=100</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_displaywidowpenalty=50</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_predisplaypenalty=10000</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_postdisplaypenalty=0</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_delimitershortfall=5pt</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_nulldelimiterspace=1.2pt</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_scriptspace=0.5pt</td>
<td>% \text{Mathspaceafter script used in} _setmathdimens, _setunimathdimens \text{instead}</td>
<td></td>
</tr>
<tr>
<td>_maxdepth=4pt</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_splitmaxdepth=_maxdimen</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_boxmaxdepth=_maxdimen</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>_parskip=0pt plus 1pt</td>
<td>% \text{Top edge of page-box to first baseline distance}</td>
<td></td>
</tr>
<tr>
<td>_abovedisplayskip=12pt plus 3pt minus 9pt</td>
<td>% \text{Above display math}</td>
<td></td>
</tr>
<tr>
<td>_abovedisplayshortskip=0pt plus 3pt</td>
<td>% \text{Above display short math}</td>
<td></td>
</tr>
<tr>
<td>_belowdisplayskip=12pt plus 3pt minus 9pt</td>
<td>% \text{Below display math}</td>
<td></td>
</tr>
<tr>
<td>_belowdisplayshortskip=7pt plus 3pt minus 4pt</td>
<td>% \text{Below display short math}</td>
<td></td>
</tr>
<tr>
<td>_parfillskip=0pt plus 1fil</td>
<td>% \text{Right margin filling}</td>
<td></td>
</tr>
<tr>
<td>_thinskipskip=3mu</td>
<td>% \text{Minimum vertical space}</td>
<td></td>
</tr>
<tr>
<td>_medskip=4mu plus 2mu minus 4mu</td>
<td>% \text{Medium vertical space}</td>
<td></td>
</tr>
<tr>
<td>_thickskip=5mu plus 5mu</td>
<td>% \text{Thick vertical space}</td>
<td></td>
</tr>
</tbody>
</table>

Note that \texttt{\_topskip} and \texttt{\_splitskip} are changed when first \texttt{\_typsize} sets the main values (default font size and default \texttt{\_baselineskip}).

### 2.7.2 Plain TeX registers

Allocate registers that are used just like in plain TeX.
\_smallskipamount, \_medskipamount, \_bigskipamount, \_normalbaselineskip, \_normallineskip, \_normallineskiplimit, \_jot, \_interdisplaylinepenalty, \_interfootnotelinepenalty.

\% 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

48
Plain TeX macros for setting parameters. \normalbaselines, \frenchspacing, \nonfrenchspacing.

\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;

2.7.3 Different settings than in plain TeX

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

The \nonfrenchspacing is not set by default because the author of OpTEX is living in Europe. If you set \enlang hyphenation patterns then \nonfrenchspacing is set.

\normalbaselines % baseline setting, 10 pt font size

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.

\emergencystretch=20pt % we want to use third pass of paragraph building algorithm
% we don't need compatibility with old documents

\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
\pdfvorigin=0pt % origin is exactly at upper left corner
\pdfhorigin=0pt
\hoffset=0pt
\voffset=0pt
\hsize=6.5in
\vsize=8.9in

If you insist on plain TeX values of these parameters then you can call the \plaintexsetting 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 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 LuaTeX 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.

More parameters accepted by \pdfximage primitive can be set in the \picparams tokens list. For example \picparams={page3} selects page 3 from included PDF file.

\kvdict is dictionary name when \readkv, \kxv, \kxv, and \iskv are processed. The default is empty.

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 \code{$\langle$text$\rangle$}. 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 parameters.opm
267 \newcount \ttline \ttline=-1 % last line number in \begtt...\endtt
268 \public \ttline ;

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.

parameters.opm
269 \newdimen \ttindent \ttindent=\parindent % indentation in verbatim
270 \newdimen \ttshift
271 \newdimen \iindent \iindent=\parindent
272 \public \ttindent \ttshift \iindent ;

The tabulator ^^I has its category code like space: it behaves as a space in normal text. This is a common plain \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.

parameters.opm
303 \newcount \tabspaces \tabspaces=3
304 \public \tabspaces ;

\hicolors can include a list of \hicolor commands with re-declarations of default colors mentioned in the \_hicolors⟨name⟩ from hisyntax-⟨name⟩.opm file. The user can give his/her preferences about colors for syntax highlighting by this tokens list.

parameters.opm
314 \newtoks \hicolors
315 \public \hicolors ;

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 \olistskipamount is vertical skip above and below the items list if \ilevel=1.
The \ilistskipamount is vertical skip above and below the items list if \ilevel>1.
The \itemskipamount is vertical skip between list items, but not above the first and below the last.

parameters.opm
336 \newtoks \defaultitem \defaultitem={$\bullet$\enspace}
337 \newtoks \everyitem
338 \newtoks \everylist
339 \newcount \i-level
340 \newskip \olistskipamount \olistskipamount=\medskipamount
341 \newskip \ilistskipamount \ilistskipamount=\medskipamount
342 \newskip \itemskipamount \itemskipamount=\medskipamount
343 \newskip \i-level
344 \public \defaultitem \everyitem \everylist \ilevel
345 \olistskipamount \ilistskipamount \itemskipamount ;
346 \let \ilistskipamount = \olistskipamount % for backward compatibility

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

parameters.opm
352 \newskip \titskip \titskip=40pt \relax % \vglue above title printed by \tit
353 \public \titskip ;

The \begmulti and \endmulti pair creates more columns. The parameter \colsep declares the space between columns. If \n columns are specified then we have \n-1 \colseps and \n 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 \tocl:(num) macro which starts printing the line.

The \bibexhook 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 2.32.6.

The \bibpart saves the name of bib-list if there are more bib-lists in single document, see section 2.32.1.

The \everyii tokens list is used before \noindent for each Index item when printing the Index.

The \everymnote is used in the \mnote group before \noindent which immediately precedes marginal note text.

The \mnotesize is the horizontal size of the marginal notes.

The \mnoteindent is horizontal space between body-text and marginal note.

The \table parameters follow. The \thistable tokens list register should be used for giving an exception for only one \table which follows. It should change locally other parameters of the \table. It is reset to an empty list after the table is printed.

The \everytable tokens list register is applied in every table. There is another difference between these two registers. The \thistable is used first, then strut and baselineskip settings are done, then \everytable is applied and then the table is printed.

\tabstrut configures the height and depth of lines in the table. You can declare \tabstrut={}, then normal baselineskip is used in the table. This can be used when you don’t use horizontal nor vertical lines in tables.

\tabiteml is applied before each item, \tabitemr is applied after each item of the table.

\tablinespace is additional vertical space between horizontal rules and the lines of the table.

\hhkern gives the space between horizontal lines if they are doubled and \vvkern gives the space between such vertical lines.

\tabskip is \tabskip used before first column, \tabskipr is \tabskip used after the last column.

\tsize is virtual unit of the width of paragraph-like table items when \table pxto\langle size\rangle is used.
The `\eqalign` macro can be configured by `\eqlines` 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.

\begin{verbatim}
\_newtoks \_eqlines \_eqlines ={\_openup\_jot}
\_newtoks \_eqstyle \_eqstyle ={\_strut\_displaystyle}
\_newdimen \_eqspace \_eqspace =20pt
\_public \eqlines \eqstyle \eqspace ;
\end{verbatim}

`\lmfil` is “left matrix filler” (for `\matrix` columns). The default value does centering because the right matrix filler is directly set to `\hfil`.

\begin{verbatim}
\_newtoks \_lmfil \_lmfil ={\_hfil}
\_public \lmfil ;
\end{verbatim}

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}`. Then use them in headline and footnote:

\begin{verbatim}
\headline={\itfixed Text of headline, section: \firstmark \hss}
\footline={\rmfixed \ifodd\pageno \hfill\fi \numprint\folio \hfil}
\end{verbatim}

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.

\begin{verbatim}
\_newdimen \_headlinedist \_headlinedist =14pt
\_newdimen \_footlinedist \_footlinedist =24pt
\_public \headlinedist \footlinedist ;
\end{verbatim}

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.

\begin{verbatim}
\_newskip \_pgbottomskip \_pgbottomskip =0pt \_relax
\_public \pgbottomskip ;
\end{verbatim}

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:

\begin{verbatim}
\headline={} \nextpages={\headline={\rmfixed \firstmark \hfil}}
\end{verbatim}

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

\begin{verbatim}
\_newtoks \_nextpages
\_public \nextpages ;
\end{verbatim}

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:
The parameters used in \in oval 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” which lists selected commands from chapter 1 and gives their behavior when a converter from OpTeX document to HTML or Markdown or B\La T\Ex 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 \cnvinfo\{⟨data⟩\}. OpTeX simply ignores this but the converter can read its configuration from here. For example, a user can write:

\cnvinfo \{type=html, \langle cnv-to-html-data \rangle\}
\cnvinfo \{type=markdown, \langle cnv-to-markdown-data \rangle\}

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.

2.8 More OpTeX macros

The second bundle of OpTeX macros is here.

We define \opinput\{⟨file name⟩\} macro which does \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 \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 \optexcatcodes catcode table and \plaintexcatcodes. They save the commonly used catcode tables. Note that \catcodetable is a part of Lua\La T\Ex extension. The catcodetable stack is implemented by OpTeX macros. The \setctable \{catcode table\} pushes current catcode table to the stack and activates catcodes from the \{catcode table\}. The \restorectable returns to the saved catcodes from the catcode table stack.

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

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

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.

The \load \langle filename-list \rangle 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 a defined macro. If the \_afterload macro is defined then it is run after \opinput. The catcode setting should be here. Note that catcode setting done in the loaded file is forgotten after the \opinput.

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

The \noexpandafter ignores the following optional space at expand processor level using the negative \romannumeral trick. The \noexpandfuturelet behaves like \futurelet primitive, but it ignores the following optional space and works at expand processor level.
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{⟨text⟩}` macro reads the text to the end of the current line and ignores it. The `\skiptoeol` macro is not expandable, but you can create your expandable macro, for example:

```
\replAB {text A \and A} % expands to "text B \and B"
```

The `\replstring\macro{(textA)}{(textB)}` replaces all occurrences of ⟨textA⟩ by ⟨textB⟩ in the macro body. The `\macro` must be defined without parameters. The occurrences of ⟨textA⟩ are not replaced if they are “hidden” in braces, for example `{⟨textA⟩}...`. The category codes in the ⟨textA⟩ must exactly match.

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

The `\replstring` macro is not expandable, but you can create your expandable macro, for example:

```
\replAB{⟨text A \and A⟩} % expands to "text B \and B"
```
The \texttt{\catcode} primitive is redefined here. Why? There is very common cases like \texttt{\catcode \{something\}} or \texttt{\catcode \number} but these characters \texttt{~} or " can be set as active (typically by \texttt{\verbchar} macro). Nothing problematic happens if re-defined \texttt{\catcode} is used in this case.

If you really need primitive \texttt{\catcode} then you can use \texttt{\_catcode}.

The \texttt{\_removespaces \texttt{\(text \ with \ spaces\)}} expands to \texttt{\(text \ without \ spaces\)}. The \texttt{\_ea \_ignorept} expands to a decimal number \texttt{\the \(dimen\)} but without pt unit.

If you do \texttt{\let\foo=a} then it is not simple to return from \texttt{\foo} to the original character code of \texttt{a}. You can write \texttt{\`a} but you cannot write \texttt{\`foo}. The macro \texttt{\cstochar \(sequence\)} solves this problem. If the sequence is equal to a character then it expands to this character (always with catcode 12). If it isn't equal to a character then it expands to nothing. You can say \texttt{\_expanded \(\cstochar\foo\)} if you want to extract the character code.

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

\begin{verbatim}
\pdfliteral{q \_bp{.3\hsize-2mm} \_bp{2mm} m 0 \_bp{-4mm} 1 S Q}
\end{verbatim}

You can use expandable \texttt{\_expr \(\langle \texttt{expression} \rangle\)} for analogical purposes. It expands to the value of the \texttt{\(\langle \texttt{expression} \rangle\)} at expand processor level. The \texttt{\(\langle \texttt{expression} \rangle\)} can include \texttt{+-*/()} and decimal numbers in common syntax. The math functions (and \texttt{\pi} constant) have to be prefixed by \texttt{\math.}, because it is processed by Lua interpreter. For example \texttt{\_expr\texttt{\(\math.pi*\math.sqrt(2)\)}}. The list of available functions is in \texttt{\_lua} manual.

You can set the number of decimal digits after decimal point of the results of \texttt{\_bp} and \texttt{\_expr} by optional syntax \texttt{\_bp \(\{\texttt{digits}\}\) \(\{\dimen\}\)} and \texttt{\_expr \(\{\texttt{digits}\}\) \(\langle \texttt{expression} \rangle\)}. Default is \texttt{\_decdigits}.

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

You can write \texttt{\_setpos \(\{\_label\}\)} somewhere and the position of such \texttt{\_setpos \(\{\_label\}\)} can be referenced by \texttt{\_posx \(\{\_label\}\)}, \texttt{\_posy \(\{\_label\}\)} and \texttt{\_pospg \(\{\_label\}\)}. The first two macros expand to \texttt{x} and \texttt{y} position measured from left-bottom corner of the page (dimen values) and \texttt{\_pospg \(\{\_label\}\)} expands to the \texttt{\_pageno}, i.e. to the page number counted from one at beginning of the document. These values are available in the second (and more) \TeX{} run, because the information is saved to \texttt{.ref} file and restored from it at the beginning of the \TeX{} job. If these values are not known then mentioned macros expand to \texttt{0sp}, \texttt{0sp} and \texttt{0}. The following example implements \texttt{\_linefrom \(\{\_label\}\)} and \texttt{\_lineto \(\{\_label\}\)} macros. The line connecting these two points is drawn (after second \TeX{} run):

\end{verbatim}
This is a text.\linefrom[A]\par
This is second paragraph with a text.\lineto[A]

Try to reverse from-to and watch the changes.

The coordinates are saved to the \_Xpos\_ref\_file in the format \_Xpos{⟨label⟩}{⟨x-pos⟩}{⟨y-pos⟩}. The \_Xpos macro defines \_pos as \{⟨x-pos⟩}{⟨y-pos⟩}{⟨total-pg⟩}{⟨rel-pg⟩}. We need to read only given parameter by \_posi, \_posii or \_posiii auxiliary macros. The implementation of \_setpos, \_posx and \_posy macros are based on \_savepos \_pdflastxpos and \_pdflastypos pdf\TeX\ primitives. The \_pospg simply reads the data from the \_currpage macro.

\begin{verbatim}
\_def\_Xpos#1#2#3{\_sxdef{\_pos:#1}{{#2}{#3}{\_currpage}}}
\_def\_setpos[#1]{{\_openref{\_pdfsavepos}{\_ewref{\_Xpos{{#1}\_unexpanded{{\_the\_pdflastxpos}{\_the\_pdflastypos}}}}}}}
\_def\_posx [#1]{\_ea \_posi \_expanded {\_trycs{\_pos:#1}{{0}{}{}{}}sp}}
\_def\_posy [#1]{\_ea \_posii \_expanded {\_trycs{\_pos:#1}{{}{}{0}{}{}}sp}}
\_def\_pospg[#1]{\_ea \_posiii \_expanded {\_trycs{\_pos:#1}{{}{}{0}{}{}}}}
\_def\_posi #1#2#3#4{#1} \_def\_posii #1#2#3#4{#2} \_def\_posiii #1#2#3#4{#3}
\_public \setpos \posx \posy \pospg ;
\end{verbatim}

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

\begin{verbatim}
\_doc ⟨ignored text⟩
⟨documentation⟩
\_cod ⟨ignored text⟩
\end{verbatim}

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

\begin{verbatim}
\_long\_def\_doc #1\_cod {\_skiptoeol}
\end{verbatim}

\docgen processes lines before \_codedecl because the version text in the macro \_⟨pkg⟩_version can be defined here. The package documentation can print it. \docgen prints banner to log because TeX doesn’t do it when command line doesn’t begin with the main file name after parameters.

\begin{verbatim}
\_long\_def\_docgen #1 {\_ea \_docgenA \_input(#1.opm)}
\_long \_def\_docgenA #1\_codedecl#2\_endcode #3\_doc {#1\_vlog{\_banner}\_skiptoeol}
\_public \docgen ;
\end{verbatim}

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:

\begin{verbatim}
\myframe [margins=5pt,rule-width=2pt,frame-color=\Red,text-color=\Blue] {text1}
or
\myframe [frame-color=\Blue] {text2} % other parameters are default
or simply \myframe {text3}. You can define \myframe as follows:
\end{verbatim}
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 read default parameters by \readkv{<values>}, and secondly the actual parameters are read by \readkv{\the\opt}. The last setting wins. Third: the values can be used by the expandable \kv<key>} macro. \kv<key>} returns \?? if such key is not declared. You can use keys without values in the parameters list too. Then you can ask if the key is declared by \iskv<key>} \iftrue or the key is undeclared by \iskv<key>} \iffalse. For example, you write to your documentation of your code that user can set the \texttt{draft} option without the value. Then you can do
\optdef\myframe [] #1{% #1 is the optional argument \readkv\myframedefaults \readkv{\the\opt} \iskv{draft} \iftrue ...draft mode... \else ...final mode... \fi}

Maybe, you want to allow not only \texttt{draft} option but \texttt{final} option (which is opposite to \texttt{draft}) too and you want to apply the result from the last given option. Then \iskv doesn't work because you can only check if both options are declared but you don't know what one is given as last. But you can use \kvx<key>}{\code} to declare \texttt{\code} which is processed immediately when the \texttt{\key} is processed by \readkv. For example
\newcount\mydraftmode
\kvx{draft}{\mydraftmode=1}
\kvx{final}{\mydraftmode=0}
\optdef\myframe [] #1{% #1 is the optional argument \readkv\myframedefaults \readkv{\the\opt} \ifnum\mydraftmode=1 ...draft mode... \else ...final mode... \fi}

The syntax of \kvx<key>}{\code} allows to use #1 inside the code. It is replaced by the actual \texttt{\value}. Example: \kvx<opt>{\message{opt is #1}}, then \readkv{opt=HELLO} prints “opt is HELLO”.

The \nokvx<code>} can declare a \texttt{\code} processed for all \texttt{\key}s undeclared by \kvx. The \#1 and \#2 can be used in the \texttt{\code}, \#1 is \texttt{\key}, \#2 is \texttt{\value}. If \nokvx is unused then nothing is done for undeclared \texttt{\key}. Example: \nokvx{\opwarning{Unknown option "#1"}}.

The default dictionary name (where key-value pairs are processed) is empty. You can use your specific dictionary by \kvdict{name}. Then \readkv, \kv, \iskv, \kvx and \nokvx macros use this named dictionary of \texttt{\key}/\texttt{\value} pairs. Package options can be processed when \kvdict{pkg:pkg}, example is the \texttt{mathset} macro in \texttt{math_opm} package.

Recommendation: If the value of the key-value pair includes = or , or }, then use the syntax \texttt{\key}=\{\texttt{\value}\}.

A more extensive example can be found in \texttt{OpTeX} trick 0073.

\begin{verbatim}
\_codedecl \readkv {Key-value dictionaries <2023-10-23>} % preloaded in format
\end{verbatim}

\textbf{Implementation.}

The \texttt{\readkv} expands its parameter and does replace-strings in order to remove spaces around equal signs and commas. Then \texttt{\kvscan} reads the parameters list finished by \texttt{\_fin} and saves values to \texttt{\kv:<dict>:\key>} macros. The \texttt{\kvx:<dict>:\key>} is processed (if it is defined) with parameter \texttt{\value}.
after it. The \kvx{key}{code} defines the \kvx{dict}{key} macro and \nokvx{code} defines the \nokvx{dict}{key} macro. The \kvx{key}{code} expands the \kv{dict}{key} macro. If this macro isn’t defined then \kvunknown is processed. You can re-define it if you want.

The \iskv{key}{true} or \iskv{key}{false} is the test, if the \kv{key} is defined in current \dict.

<table>
<thead>
<tr>
<th>Plain Te\LaTeX\ macros</th>
</tr>
</thead>
<tbody>
<tr>
<td>All macros from plain Te\LaTeX are rewritten here. Differences are mentioned in the documentation below.</td>
</tr>
</tbody>
</table>

The \dospecials works like in plain Te\LaTeX 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 OpTeX. Use normal numbers instead of such obscurities.

Plain \TeX basic macros and control sequences. \endgraf, \endline. The \^^L is not defined in OpTeX because it is obsolete.

Plain \TeX classical \obeylines and \obeyspaces.

% in \obeylines, we say `\let\^^L=\_par' instead of `\def\^^L{\_par}'}
| \% since this allows, for example, `\let\par\cr \obeylines \halign{'...} |
| \% these lines must end with |
| \% this is in case \^^L appears in a \write |
| \% these lines must end with |

60
Spaces. \thinspace, \negthinspace, \enspace, \enskip, \quad, \qquad, \smallskip, \medskip, \bigskip, \nointerlineskip, \offinterlineskip, \topglue, \vglue, \hglue, \slash.

```
\protected\def\thinspace{\kern.16667em}
\protected\def\negthinspace{\kern-.16667em}
\protected\def\enspace{\kern.5em}
\protected\def\enskip{\hskip.5em\relax}
\protected\def\quad{\hskip1em\relax}
\protected\def\qquad{\hskip2em\relax}
\protected\def\smallskip{\vskip\smallskipamount}
\protected\def\medskip{\vskip\medskipamount}
\protected\def\bigskip{\vskip\bigskipamount}
\protected\def\nointerlineskip{\prevdepth=-1000pt}
\protected\def\offinterlineskip{\baselineskip=-1000pt \lineskip=0pt \lineskiplimit=\maxdimen}
\protected\def\topglue{\nointerlineskip\vglue-\topskip\vglue} % for top of page
\protected\def\vglue{\afterassignment\vglA\skip0=}
\protected\def\vglA{\par\dimen0=\prevdepth\hrule height0pt\nobreak\vskip\skip0\prevdepth=\dimen0}
\protected\def\hglue{\afterassignment\hglA\skip0=}
\protected\def\hglA{\leavevmode\count255=\spacefactor\vrule width0pt\nobreak\hskip\skip0\spacefactor=\count255}
\protected\def\~{\penalty10000 \ }
\protected\def\slash{/\penalty\exhyphenpenalty}
```

Penalties macros: \break, \nobreak, \allowbreak, \filbreak, \goodbreak, \eject, \supereject, \dosupereject, \removelastskip, \smallbreak, \medbreak, \bigbreak.

```
\protected\def\break{\penalty-10000}
\protected\def\nobreak{\penalty10000}
\protected\def\allowbreak{\penalty0}
\protected\def\filbreak{\par\fil\penalty-200\vfilneg}
\protected\def\goodbreak{\par\penalty-500}
\protected\def\eject{\par\break}
\protected\def\supereject{\par\penalty-20000}
\protected\def\dosupereject{\ifnum\insertpenalties>0 % something is being held over
\par\fil\penalty-100\vfilneg\ifvmode\penalty0\vfil\fi\fi}
\protected\def\removelastskip{\ifdim\lastskip=\zo\else\vskip-\lastskip\fi}
\protected\def\smallbreak{\par\ifdim\lastskip<\smallskipamount\removelastskip\penalty-50\smallskip\fi}
\protected\def\medbreak{\par\ifdim\lastskip<\medskipamount\removelastskip\penalty-100\medskip\fi}
\protected\def\bigbreak{\par\ifdim\lastskip<\bigskipamount\removelastskip\penalty-200\bigskip\fi}
```

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

```
\protected\def\line{\hbox to\hsize}
\protected\def\leftline#1{\line{#1\hss}}
\protected\def\rightline#1{\line{#1\hss}}
\protected\def\centerline#1{\line{#1\hss}}
\protected\def\rlap#1{\hbox to\zo{#1\hss}}
\protected\def\llap#1{\hbox to\zo{#1\hss}}
\protected\def\underbar#1{\ifdim\netbox=0\hbox to\zo{\dop=\zo \math \underline{\box0}}}\fi}
```

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

Tabbing macros are omitted because they are obsolete.

Indentation and others. \texttt{\textindent, \item, \itemitem, \narrower, \raggedright, \tt\raggedright, \leavevmode}.

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 \texttt{\unichars} macro is run in \texttt{\initunifonts}, Unicodes are used instead old plain TeX settings.
Accents. The macros \oalign, \d, \b, \c, \dots, are defined for backward compatibility.

The accent commands like \v, \., \H, etc. are not defined. Use the accented characters directly – it is the best solution. But you can use the macro \oldaccents which defines accented macros. Much more usable is to define these control sequences for other purposes.
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.

\public \hrulefill \dotfill \rightarrowfill \leftarrowfill \downbracefill \upbracefill;

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

\public \showhyphens \bye;

Plain TeX reads hyphen.tex with patterns as \language=0. We do the same.

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 of 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...
8-bit 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.

If the *.tfm files are missing during format generation then the format is successfully generated
without any pre-loaded fonts. It doesn’t matter if each document processed by OpTEX declares Uni-
code fonts. You can create such fonts-less format anyway if you set \fontspreload to \relax before
\input optex.ini, i.e.: luatex -ini '{\let\fontspreload=\relax \input optex.ini'}

2.12 Using \font primitive directly

You can declare a new font switch by \font primitive:

\font{⟨font switch⟩} = ⟨font file name⟩ ⟨size spec⟩
% for example:
\font{\tipa} = tipa10 at12pt % the font tipa10 at 12pt is loaded
% usage:
{\tipa TEXT} % the TEXT is printed in the loaded font.

The ⟨size spec⟩ can be empty or at⟨dimen⟩ or scaled⟨scale factor⟩. The ⟨font file name⟩ must be termi-
nated by space or surrounded in the braces.

OpTEX starts with \font primitive which is able to read only tfm files, i.e. the ⟨font file name⟩.tfm
(and additional data for glyphs) must be correctly installed in your system. If you want to load OpenType
otf or ttf font files, use the declarator \initunifonts before first \font primitive. This command adds
additional features to the \font primitive which gives the extended syntax:

\font{⟨font switch⟩} = {⟨font file name⟩:⟨font features⟩} ⟨size spec⟩
% or
\font{⟨font switch⟩} = {⟨font name⟩:⟨font features⟩} ⟨size spec⟩

where ⟨font file name⟩ is name of the OpenType font file with the extension .otf or .ttf or without
it. The braces in the syntax are optional, use them when the ⟨font file name⟩ or ⟨font name⟩ includes
spaces. The original syntax for tfm files is also available. Example:

\initunifonts
\font\crimson=[Crimson-Roman] at11pt % the font Crimson-Regular.otf is loaded
\font\crimsonff=[Crimson-Roman]:+smcp:+onum at11pt % The same font is re-loaded
% with font features
{\crimson Text 12345} % normal text in Crimson-Regular
{\crimsonff Text 12345} % Crimson-Regular with small capitals and old digits

\initunifonts loads the implementation of the \font primitive from luatfload package. More inform-
information is available in the luatfload-latex.pdf file.
You can use \texttt{\ufont} macro which runs \texttt{\initunifonts} followed by \texttt{\font} primitive. And \texttt{\fontfam} does (among other things) \texttt{\initunifonts} too. You need not to specify \texttt{\initunifonts} if \texttt{\fontfam} or \texttt{\ufont} is used.

When \texttt{\initunifonts} is declared then the \texttt{\font} primitive is ready to read Type1 fonts too. If you have \texttt{file.afm} and \texttt{file.pfb} then you can declare \texttt{\font\f=file.afm} and use \texttt{\f}. It means that you needn’t to create \texttt{tfm} files nor \texttt{vf} files, you can use Type1 fonts directly. They behave as Unicode fonts if the \texttt{afm} metrics are implemented correctly (with correct names of all included glyphs). But we must to say that Type1 font format is old technology, the loading of Type1 fonts is not optimized. Use OpenType fonts (\texttt{otf} or \texttt{ttf}) if it is possible.

Let’s sum it up. Suppose that \texttt{\initunifonts} was used. The \texttt{\font} primitive is able to load OpenType fonts (\texttt{otf} or \texttt{ttf}), Type1 fonts (\texttt{afm} and \texttt{pfb}) or classical \texttt{tfm} fonts. We strongly recommend to prefer OpenType format over Type1 format over \texttt{tfm} format. The last one doesn’t support Unicode. If there is nothing else left and you must to use \texttt{tfm}, then you must to implement re-encoding from Unicode to the \texttt{tfm} encoding at macro level, see the \texttt{Op\TeX} trick 0018 for example.

### 2.12.1 The \texttt{\setfontsize} macro

It seems that you must decide about final size of the font before it is loaded by the \texttt{\font} primitive. It is not exactly true; \texttt{Op\TeX} offers powerful possibility to resize the font already loaded on demand.

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

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

The initial value in \texttt{Op\TeX} is given by \texttt{\setfontsize\{at10pt\}}.

The \texttt{\resizethefont} resizes the currently selected font to the size given by previous \texttt{\setfontsize} command. For example:

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

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

The following example demonstrates the \texttt{mag} format of \texttt{\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.
```

The \texttt{\resizethefont} works with arbitrary current font, for example with the font loaded directly by \texttt{\font} primitive. For example:

```
\ufont\tencrimson=[Crimson-Roman]:+onum % font Crimson-Regular at 10 pt is loaded
\def\crimson{\tencrimson\resizethefont} % \crimson uses the font size on demand
\crimson The 10 pt text is here.
\setfontsize\{at12pt\}
\crimson The 12 pt text is here.
```

This is not only an academical example. The \texttt{\csrimson} command defined here behaves like variant selector in the Font Selection System (section 2.13). It takes only information about size from the font context, but it is sufficient. You can use it in titles, footnotes, etc. The font size depending on surrounding size is automatically selected. There is a shortcut \texttt{\sfont} with the same syntax like \texttt{\font} primitive, it declares a macro which selects the font and does resizing depending on the current size. So, the example above can be realized by \texttt{\sfont\crimson=[Crimson-Roman]:+onum}.
2.12.2 The \font-like commands summary

- \font is TeX primitive. When OpTEX starts, then it accepts only classical TeX syntax and doesn’t allow to load Unicode fonts. Once \initunifonts (or \fontfam) is used, the \font primitive is re-initialized: now it accepts extended syntax and it is able to load Unicode OpenType fonts.
- \ufont is a shortcut of \initunifonts \font. I.e. it behaves like \font and accepts extended syntax immediately.
- \sfont has syntax like extended \font. It declares a macro which selects the given font and resizes it to the current size (given by \setfontsize). In various part of document (text, footnotes, titles), the size of this font is selected by the declared macro properly.

2.12.3 The \fontlet declarator

We have another command for scaling: \fontlet which can resize arbitrary font given by its font switch.

\fontlet \langle new font switch \rangle = \langle given font switch \rangle \langle size spec \rangle

example:
\fontlet \bigfont = \_tenbf at15pt

The \langle given font switch \rangle must be declared previously by \font or \fontlet or \fontdef. The \langle new font switch \rangle is declared as the same font at given \langle size spec \rangle. The equal sign in the syntax is optional. You can declare \langle new font switch \rangle as the scaled current font by

\fontlet \langle new font switch \rangle = \font \langle size spec \rangle

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”. Each design size is implemented in its individual font file and OpTEX is able to choose right file if various optical sizes and corresponding file names are declared for the font by \_regtfm or \_regoptfsize command. The command \setfontsize sets the internal requirements for optical size if the parameter is in the format at\langle dimen\rangle or mag\langle factor\rangle. Then the command \resizethefont or \fontlet or variant selectors try to choose the font suitable for the required optical size. For example

\fontfam[lm]
\setfontsize{at13pt}rm
\resizethefont
Now, the text is printed in \[lmroman12-regular\] at 13 pt.

See also section 2.13.12.

2.12.5 Font rendering

If \initunifonts isn’t declared then OpTEX uses classical font renderer (like in pdftex). The extended font renderer implemented in the Luaotfload package is started after \initunifonts.

The OpTEX format uses luatex engine by default but you can initialize it by luahbtex 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.

2.12.6 Implementation of resizing

Only “resizing” macros and \initunifonts are implemented here. Other aspects of Font Selection System and their implementation are described in section 2.13.14.

\setfontsize \langle Font resizing macros <2022-11-08> \rangle % preloaded in format
\\initunifonts macro extends LuaTeX’s font capabilities, in order to be able to load Unicode fonts. Unfortunately, this part of OpTEX depends on the luaotfload package, which adapts ConTeXt’s generic
font loader for plain \TeX{} and \LaTeX. \texttt{luaotfload} uses Lua functions from \LaTeX{}’s \texttt{luatexbase} namespace, we provide our own replacements. \texttt{\textbackslash initunifonts} sets itself to relax because we don’t want to do this work twice. \texttt{\textbackslash ufont} is a shortcut of \texttt{\textbackslash initunifonts \font}.

\begin{verbatim}
\protected\def \initunifonts {}% \directlua{require('luaotfload-main') luaotfload.main() \optex.hook_into_luaotfload()}% \_glet \_fmodtt=\_unifmodtt % use \_ttunifont for \tt \_glet \initunifonts=\_relax % we need not to do this work twice \_glet \initunifonts=} \_relax\_protected\def \ufont {\initunifonts \font} \_protected\def \ufont \font
\end{verbatim}

The \texttt{\textbackslash setfontsize \{size spec\}} saves the \texttt{\{size spec\}} to the \texttt{\_sizespec} macro. The \texttt{\_optsize} value is calculated from the \texttt{\{size spec\}}. If the \texttt{\{size spec\}} is in the format \texttt{scaled\langle factor\rangle} then \texttt{\_optsize} is set from \texttt{\defaultoptsize}. If the \texttt{\{size spec\}} is in the \texttt{mag\langle number\rangle} format then the contents of the \texttt{\_sizespec} macro is re-calculated to the \texttt{at\langle dimen\rangle} format using previous \texttt{\_optsize} value.

\begin{verbatim}
\newdimen \optsize \optsize=10pt \newdimen \defaultoptsize \defaultoptsize=10pt \newdimen \lastmagsize \def \setfontsize #1{\edef \sizespec {#1} \setoptsize \sizespec \_relax} \def \setoptsize {\isnextchar a { \setoptsizeA } { \isnextchar m { \setoptsizeB } { \setoptsizeC } } } \def \setoptsizeA at#1 \_relax { \optsize=#1 \_lastmagsize=\optsize } % at\langle dimen\rangle \def \setoptsizeB scaled#1 \_relax { \optsize=\defaultoptsize \_relax } % scaled\langle scalenum\rangle \def \setoptsizeC mag#1 \_relax { % scaled\langle scalenum\rangle \ifdim \lastmagsize > \zo \_optsize=\lastmagsize \else \_optsize=\pdffontsizesize \font \_fi \_optsize=\optsize \_lastmagsize=\optsize \edef \sizespec { at \_the \_optsize } } \def \setfontsize \defaultoptsize
\end{verbatim}

The \texttt{\textbackslash fontname} primitive returns the \texttt{\{font file name\}} optionally followed by \texttt{\{size spec\}}. The \texttt{\xfontname} macro expands to \texttt{\{font file name\}} without \texttt{\{size spec\}}. We need to remove the part \texttt{\{space\}at\langle dimen\rangle} from \texttt{\fontname} output. The letters \texttt{at} have category 12.

\begin{verbatim}
\edef \stringat { \string a \string t } \edef \xfontname #1 { \unexpanded { \_ea \xfontnameA \fontname \stringat } \_relax } \expanded { \expanded { \_def \noexpand \xfontnameA #1 \_stringat #1 \_relax } #1 } \edef \fontlet \langle font switch A \rangle \langle font switch B \rangle \{ size spec \} does \font \langle font switch A \rangle = \langle \{ font file name \} \rangle \langle size spec \rangle
\end{verbatim}

Note, that the \texttt{\xfontname} output is converted due to optical size data using \texttt{\_optfn}.

\begin{verbatim}
\protected\def \fontlet \#1\#2{\_ifx \#2=\_ea \fontlet \#1 \else \_ea \font \_ea \#1 \_expanded { \_optfn { \xfontname \#2 } } \_fi} \public \xfontname \fontlet ;
\end{verbatim}

\texttt{\newcurrfontsize \{size spec\}} does \texttt{\fontlet \{saved switch\}=\font \{size spec\} \_relax \{saved switch\}}. It changes the current font at the given \texttt{\{size spec\}}. \texttt{\resizethefont} is implemented by \texttt{\newcurrfontsize} using data from the \texttt{\_sizespec} macro. \texttt{\sfont} has the same syntax like \texttt{\font} primitive, but declares a macro which selects the font and sets its size properly dependent on the current size.
Optical sizes data for preloaded 8bit Latin Modern fonts:

\[\text{\regtfm} \text{lmr} 0 \text{ec-lmr5 5.5 ec-lmr6 6.5 ec-lmr7 7.5 ec-lmr8 8.5 ec-lmr9 9.5} \]
\[\text{ec-lmr10 11.1 ec-lmr12 15 ec-lmr17} \]
\[\text{\regtfm} \text{lmbx} 0 \text{ec-lmbx5 5.5 ec-lmbx6 6.5 ec-lmbx7 7.5 ec-lmbx8 8.5 ec-lmbx9 9.5} \]
\[\text{ec-lmbx10 11.1 ec-lmbx12} \]
\[\text{\regtfm} \text{lmtt} 0 \text{ec-lmtt8 8.5 ec-lmtt9 9.5 ec-lmtt10 11.1 ec-lmtt12} \]

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 switches**, 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, 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 name 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.opm. All families are collected in fams-ini.opm and users can give more declarations in the file fams-local.opm.

### 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.

When \fontfam [(Font Family)] is used and the given family isn’t found in the current \TeX system and the (Font Family) is previously declared by \fontfamsub[(Font Family)] [(Other Family)] then OpTeX does the given substitution and runs \fontfam[(Other Family)].

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{itemize}
  \item \textbf{family selector} \textbf{font modifiers} \textbf{variant selector}
\end{itemize}

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 {⟨size spec⟩} command in the same sense as other font modifiers. It saves information about font size to the font context. See section 2.12.1. 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` ({`\fontfam` `\fontname`}) 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) 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 2.12.1), `\setff` (see section 2.13.9), `\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`, `\_famdecl` and `\def` make local assignment.

2.13.5 The \fontdef declarator in detail

You can declare `{\font-def}` by the `\fontdef` command.

```
\fontdef{\font-switch} {\family-selector} {\font-modifiers} {\variant-selector}
```

where `{\family-selector}` and `{\font-modifiers}` are optional and `{\variant-selector}` is mandatory.

The resulting `{\font-switch}` 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 `{\family-selector}` and/or `{\font-modifiers}` and it finds the real font by `{\variant-selector}`. This font is not selected but it is assigned to the declared `{\font-switch}` (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{\new-variant-selector} {\family-selector} {\font-modifiers} {\variant-selector}
```

where `{\family-selector}` and `{\font-modifiers}` are optional and `{\variant-selector}` is mandatory. The `{\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 `{\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 \langle family selector \rangle and/or \langle font modifiers \rangle, runs following \langle variant selector \rangle. 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 \langle new variant selector \rangle family dependent. When the selector is used in another family context than it is defined then a warning is printed on the terminal “\langle ear selector \rangle 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 \langle family selector \rangle can be written before \langle font modifiers \rangle in the \famvardef parameter. Then the \langle new variant selector \rangle is declared in the current family but it can use fonts from another family represented by the \langle family selector \rangle.

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

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

The variant selectors (declared by \famvardef) or font modifiers (declared by \moddef) are (typically) control sequences in the public namespace (\mf, \caps). They are most often declared in font family files and they are loaded by \fontfam. A conflict with such names in the public 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:

\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.

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).

The behavior of \tt can be re-defined by \famvardef. For example:

\fontfam[Cursor]
\fontfam[Heros]
\fontfam[Termes]
\famvardef\tt{|Cursor|\setff{-liga;-tlig}\rm}

Test in Termes: {\tt text}. {\Heros\rm Test in Heros: {\tt text}}. Test in URL \url{http://something.org}.
You can see that \texttt{tt} stay family independent. This is a special feature only for \texttt{tt} selector. New definitions of \texttt{tfont} and \texttt{urlfont} are done too. It is recommended to use \texttt{setff{-liga;-lig}} to suppress the ligatures in typewriter fonts.

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

### 2.13.8 Font commands defined by \texttt{\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 \texttt{{\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}

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

\texttt{Op\TeX} defines similar internal commands \texttt{\titfont, \chapfont, \seffont and \seccfont}, see section 2.26. The commands \texttt{\typsize} and \texttt{\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 \texttt{otfinfo -f font.otf}. For example, LinLibertine fonts provide \texttt{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 \texttt{\setff \{\feature\}} acts like family independent font modifier and prepares a new \texttt{\feature}. You must use a variant selector in order to reinitialize the font with the new font feature. For example \texttt{\setff{+frac}\rm} or \texttt{\setff{+frac}\currvar}.

You can declare a new variant selector too:

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

Compare 1/2 or 1/10 \texttt{\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 \texttt{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{verbatim}
\setfontsize{\{size spec\}} % sets the font size
\setff{\{font feature\}} % adds the font feature
\setletterspace{\{number\}} % sets letter spacing
\setwordspace{\{scaling\}} % modifies word spacing
\end{verbatim}

The \texttt{\setfontsize} command is described in the section 2.12.1. The \texttt{\setff} command was described in previous subsection.

\texttt{\setletterspace \{\{number\}\}} specifies the letter spacing of the font. The \texttt{\{number\}} 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{\{number\}} means no letter spacing which is the default.

\texttt{\setwordspace \{\{number\}\}} scales the default interword space (defined in the font) and its stretching and shrinking parameters by given \texttt{\{scaling\}} 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{\{default\}/\{stretching\}/\{shrinking\}}. For example, \texttt{\setwordspace{1.25/1}} enlarges only stretching 2.5 times.

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

```latex
\_famdecl [Heros] \{TeX Gyre Heros fonts based on Helvetica\}
{\caps \cond \(\textbf{\textit{\bi}}\) FiraMath}
{\{texgyreheros-regular\}}
{\def\fontnamegen{\{texgyreheros\}_\condV-\_currV}:\capsV\_fontfeatures}}
\_wlog{\detokenize{
Modifiers:"J
\caps \ldots \caps \& small caps"J
\cond \ldots \condensed variants"J
}}
\_moddef \resetmod {\_fsetV caps={},\_fsetV cond=\textbf{\_cond}} \_fvars regular bold italic bolditalic
\_moddef \caps {\_fsetV caps=+smcp;\_ffonum; }
\_moddef \nocaps {\_fsetV caps={}}
\_moddef \cond {\_fsetV cond=\textbf{\_cond}}
\_moddef \nocond {\_fsetV cond=\textbf{\_cond}}
\_initfontfamily \% new font family must be initialized
\_ifmathloading
\_loadmath \{FiraMath-Regular\}
\addmathfont \_xits \{XITSMath-Regular\} \{XITSMath-Bold\}{}{}
\_addto\_frak{\_xits} \_addto\_cal{\_xits} \_public \frak \cal ;
\% \texttt{bf, \textit{bf} from FiraMath:}
\_let\_bsansvariables=\_bfvariables
\_let\_bsansGreek=\_bfGreek
\_let\_bsansdigits=\_bfdigits
\_let\_bisansvariables=\_bivariables
\_let\_bisansGreek=\_bigreek
\% \texttt{\_resethchars <family-name> <list of \texttt{\umathchardef csnames}> ;}
\_mathchars \_xits \{\texttt{granule} \texttt{smallgranule} \texttt{biggranule} \texttt{smallbiggranule}\}
\% \texttt{vartriangle \texttt{smallvartriangle} \texttt{righttriangle} \texttt{smallrighttriangle}}
\_unicodevdotss \_unicodeadotss \_unicodeddotss \_unicodevdotss \% \ldots etc. you can add more
\_fi
```

If you want to write such a font family file, you need to keep the following rules.

- Use the \texttt{\_famdecl} command first. It has the following syntax:

  ```latex
  \_famdecl [(Name of family)] (Familyselector) (\{}(comments)\}\{\{}(modifiers)\}\{\{}(variant selectors)\}\{\{}(comments about math fonts)\}\{\{}(font-for-testing)\}\{\def\fontnamegen{(font name or font file name generated)}\}
  ```

  This writes information about font family at the terminal and prevents loading such file twice. Moreover, it probes existence of \texttt{(font-for-testing)} in your system. If it doesn’t exist, the file loading
is skipped with a warning on the terminal. The \_ifexistfam 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 f-\lmfonts.opm 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 check: 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 \{(font-switch) \{\_fontnamegen\} \_sizespec

Note that the extended \font syntax \font\{(font-switch) \{(font name)\;\{font features\}\} \{size spec.\} or \font\{(font-switch) \{\{(font file name)\}\;\{font features\}\} \{size spec.\} is expected here.

Example 1
Assume an abstract font family with fonts xx-Regular.otf, xx-Bold.otf, xx-Italic.otf and xx-BoldItalic.otf. Then you can declare the \_resetmod (for initializing the family) by:

\_moddef\resetmod{\_fvars Regular Bold Italic BoldItalic }\_famdecl ...
{\def\_fontnamegen{\[xx-\_currV\]}}

The following auxiliary macros are used here:

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

Assume that the user needs \it variant in this family. Then the \_fontnamegen macro expands to \[xx-\_currV\] and it expands to \[xx-Italic\]. The Font Selection System uses \font \{\[xx-Italic\]\}. This command loads the xx-Italic.otf font file.

See more advanced examples are in f-\langle family \rangle.opm files.

Example 2
The f-\heros.opm is listed here. Look at it. When Heros family is selected and \bf is asked then \font \{\{texgyreheroscn-bold\}\;+tlig;\} at10pt is processed.

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

\_fsetV \langle word-a\rangle=\langle value-a\rangle, \langle word-b\rangle=\langle value-b\rangle ... etc. terminated by a space

with obvious result \texttt{\def\langle word-a\rangle V \{\langle value-a\}\} \def\langle word-b\rangle V \{\langle value-b\}\} etc.

Example 3
If both font modifiers \caps, \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 \bf variant at 11pt now then the

\font \{\{texgyreheroscn-bold\}\;+smcp;+onum;+pnum;+tlig;\} at11pt

is processed. We assume that a font file texgyreheroscn-bold.otf is present in your \TeX system.
The \_onlyif macro has the syntax \_onlyif \langle word\rangle=\langle value-a\rangle, \langle value-b\rangle, ... , \langle value-n\rangle: \langle what\rangle. It can be used inside \moddef as simple IF statement: the \langle what\rangle is processed only if \langle word\rangle has \langle value-a\rangle or \langle value-b\rangle ... or \langle value-n\rangle. See f-roboto.opm for examples of usage of many \_onlyif's.

Recommendation: use the \_fontfeatures macro at the end of the \_fontnamegen macro in order to the \setff, \setfontcolor, \setletterstretch macros can work.

The \_moddef macro has the syntax \_moddef\langle modifier\rangle{\langle what to do\rangle}. It does more things than simple \_def:

- The modifier macros are defined as \_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 \_famvardef macro has the same features.

The \langle Familyselector\rangle is defined by the \_famdecl macro as:

```
\protected\def\langle Familyselector\rangle {%
  \_def\_currffamily \langle Familyselector\rangle%
  \_def\_fontnamegen {...}% this is copied from 7-th parameter of \_famdecl
  \resetmod
  \langle run all family-dependent font modifiers used before Familyselector without warnings\rangle}
```

The \_initfontfamily must be run after modifier's declaration. It runs the \langle Familyselector\rangle 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 \langle Familyselector\rangle only in public 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 OpTEX warns about it. See the end of section 2.13.4.

If you need to use an internal control sequence declared in your fontfile, use the reserved name space with names starting with two _ followed by family indentifier or by vf if it relates to variable fonts.

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

Please, look at OpTEX 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:

```
\fontfam [Termes] \fontfam[Heros]
\caps\cond\it Caps+Cond italic in Heros \Termes\currvar Caps italic in Termes.
```

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 OpTEX.

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 the example of the optical sizes declaration including documentation about it.

Several fonts don’t switch to the font features if the features are specified directly as documented above. You must add the script=latn; specification to the features string when using these fonts, see f-baskerville.opm for example. The reason: these fonts don’t follow the OpenType specification and
they don’t set the DFLT script but only scripts with given names like latn. And the tables implementing all
font features are included here. You can check the internals of the font by FontForge: View / Show ATT
/ OpenType Tables / GSUB. Do you see the DFLT script here?

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 \{internal-template\} and its output is a part of the font file name \{size-dependent-template\}
with respect to given optical size.

You can declare a collection of \{size-dependent-template\}s for one given \{internal-template\} by the
\_regoptsizes macro. The syntax is shown for one real case:

\_regoptsizes lmr.r lmrroman?-regular
5 <5.5 6 <6.5 7 <7.5 8 <8.5 9 <9.5 10 <11.1 12 <15 17 <*

In general:

\_regoptsizes \{internal-template\} \{general-output-template\} \{resizing-data\}

Suppose our example above. Then \_optname(1mr.r) expands to lmrroman?-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 \{resizing-data\}
virtually begins with zero <0, but it is not explicitly written. The right part of \{resizing-data\} must be
terminated by <*> which means "less than infinity".

If \_optname gets an argument which is not registered \{internal-template\} 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.om. It sets the file names to lowercase and
enables us to use shortcuts instead of real \{resizing-data\}. There are shortcuts \_regoptFS, \_regoptT,
etc. here. The collection of \{internal-templates\} are declared, each of them covers a collection of real file
names.

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

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 \{internal-templates\} and do an alias from each \{internal-template\} to \{real-font-name\}. For example, you can do it as follows:

\def\fontalias #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 \_size font modifier:

\f-poltawski.om
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-{famname}.opm and \TeX{} can see it in your filesystem then you can type \fontfam{famname} and the file is read, so the information about the font family is loaded. The name (famname) must be lowercase and without spaces in the file name f-{famname}.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 \fontfam[catalog] is printed. The list of families taken in the catalog and listed on the terminal is declared in two files: fams-ini.opm and 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 fams-ini.opm.

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

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

\\fams-ini.opm
The \faminfo command has the syntax:

```
\faminfo [\{Family Name\}] {\{comments\}} {\{file-name\}}
```

The \famtext command 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.

### 2.13.14 Implementation of the Font Selection System

The main principle of the Font Selection System is: run one or more modifiers followed by \fontsel. Modifiers save data and \fontsel selects the font considering saved data. Each basic variant selector \rm, \bf, \it, \bi, and \tt runs internal variant modifier \fmodrm, \fmodbf, \fmodit, \fmodbi and \fmodtt. These modifiers save their data to the \famv macro which is \rm or \bf or \it or \bi or \tt.

The \currvar selector is \fontsel by default, but variant selectors declared by \famvardef change it.

The \fontsel creates the \{font switch\} in the format \{ten\famv\} and loads the font associated to the \{font switch\}. The loading is done by:

a) \letfont \{font switch\} = \savedswitch \sizespec

b) \font \{font switch\} = \fontnamegen \sizespec

The a) variant is used when \fontnamegen isn’t defined, i.e. \fontfam wasn’t used: only basic variant and \sizespec is taken into account. The b) variant is processed when \fontfam was used: all data saved by all font modifiers are used during expansion of \fontnamegen.

After the font is loaded, final job is done by \fontselA \{font-switch\}.
\_else \% \fontfam is used
\_ea \% \csname _ten\_xfamv\_endcsname \_fontnamegen\_sizespec
\_fi \_relax
\_ea \% \fontselA \csname _ten\_xfamv\_endcsname
\}
\def\fontselA #1{\%
\protected\def \currvar {\fontsel}% default value of \currvar
\logfont #1% font selecting should be logged.
\setup #1% wordspace setting
\fontloaded #1% initial settings if font is loaded firstly
#1% select the font
}\}
\def \logfont #1{}
\def \xfamv {\famv}
\public \fontsel ;

If a font is loaded by macros \fontsel 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. The \mte.opm package uses it. See also Op\TeX trick 0058.

\def\fontloaded #1{\ifnum\skewchar#1=-1 \skewchar#1=-2 \newfontloaded#1\fi}
\def\newfontloaded #1{}
\ttunifont is default font for \tt variant when \initunifonts is declared. User can re-define it or use \famvardef{tt}. The \unifmodtt macro is used instead \fmodtt after \initunifonts. It ignores the loading part of the following \fontsel and do loading itself.
\_ttunifont{[lmmono10-regular];\fontfeatures-tlig;}
\_def \unifmodtt\fontsel{% ignore following \fontsel
\_ea \font \csname _ten\ttfamv\_endcsname \ttunifont \_sizespec \_relax
\_ea \fontselA \csname _ten\ttfamv\_endcsname
\_def \currvar\_tt}
\_def \ttfamv{tt}

A large part of the Font Selection System was re-implemented in Feb. 2022. We want to keep backward compatibility:

\_ttunifont{[lmmono10-regular];\fontfeatures-tlig;}
\_def \unifmodtt\fontsel{% ignore following \fontsel
\_ea \font \csname _ten\ttfamv\_endcsname \ttunifont \_sizespec \_relax
\_ea \fontselA \csname _ten\ttfamv\_endcsname
\_def \currvar\_tt}
\_def \ttfamv{tt}

The \\_famdecl \[\langle Family Name\rangle\] \\{\langle Famselector\rangle\} \\{\langle comment\rangle\} \\{\langle modifiers\rangle\} \\{\langle variants\rangle\} \\{\langle math\rangle\} \\{\langle font for testing\rangle\} \\{\langle def \fontnamegen\{\langle data\}\}\} runs \initunifonts, then checks if \\langle Famselector\rangle is defined. If it is true, then closes the file by \endinput. Else it defines \\langle Famselector\rangle and saves it to the internal \_f:\langle currfamily\rangle:main.fam command. The macro \\_initfontfamily needs it. The \\_currfamily is set to the \\langle Famselector\rangle because the following \moddef commands need to be in the right font family context. The \\_currfamily is set to the \\langle Famselector\rangle by the \\langle Famselector\rangle too, because \\langle Famselector\rangle 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.
reinitializing the same current modifiers in the font context after the family is changed.

\resetmod when a new family is selected by a family selector, see is printed if value is changed then we can declare the same \_f:\_currfamily:

and the spect the family context. In fact, \moddef

\_prepcommalist ab,{},cd,\_fin, ⟨as
⟨\_onlyif \_fsetV \langle
\_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. \_fsetV (key)=(value), \ldots, (key)=(value) expands to \def\_{key}V{(value)} in the loop. \_onlyif (key)=(value-a), (value-b), \ldots, (value-z): \{what\} runs \{what\} only if the \_{key}V is defined as (value-a) or (value-b) or ... or (value-z).

\_prepcommalist ab,{},cd,\_fin, expands to ab,,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=\_onum if you don’t want to set old digits together with \caps.

\_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. \_fsetV (key)=(value), \ldots, (key)=(value) expands to \def\_{key}V{(value)} in the loop. \_onlyif (key)=(value-a), (value-b), \ldots, (value-z): \{what\} runs \{what\} only if the \_{key}V is defined as (value-a) or (value-b) or ... or (value-z).

\_prepcommalist ab,{},cd,\_fin, expands to ab,,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=\_onum 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 \_f:\_currfamily:\{modifier\} value if it is defined or it prints the warning. When the \_currfamily value is changed then we can declare the same \{modifier\} with a different meaning.

\setnewmeaning \{cs-name\}=\_tmpa \{by-what\} does exactly \let \{csname\}=\_tmpa but warning is printed if \{cs-name\} is defined already and it is not a variant selector or font modifier.

\addtomodlist (font modifier) adds given modifier to \_modlist macro. This list is used after \resetmod when a new family is selected by a family selector, see \_resetfam macro. This allows reinitializing the same current modifiers in the font context after the family is changed.
\protected \def \resetmod \cs{f:\currfamily:resetmod} % private variant of \resetmod
\def \resetfam{%
def addtomodlist##1{} \resetmod
edef \modlist{\ea} \modlist
let addtomodlistb = addtomodlistb
ifcsname _f:\currfamily:\ea \currvar \endcsname
else \ea ifx \currvar \tt else \def \currvar{\fontsel} \fi
fi % corrected \currvar in the new family
}
def \currfamily{} % default current family is empty
def \modlist{} % list of currently used modifiers
def addtomodlist#1{addto_modlist#1}
let addtomodlistb = addtomodlist

\def famdepend#1#2{%
ifcsname#2 \endcsname \csname#2\ea \endcsname else
ifx addtomodlist addtomodlistb
opwarning{\string#1 is undeclared in family \"\currfamily\", ignored} \fi \fi
}
def setnewmeaning #1=#2{%
if #1 undefined else if #1 #2 else
opwarning{\string#1 is redefined by \string#2} %
fi \fi
let #1 = #2
}
def fontdef \fontswitch \{\data\} does:
\begingroup \data \ea \let \ea \fontswitch \the font
\endgroup \let \ea \fontswitch \the font \def \currvar{\fontswitch}

\begingroup \data \ea \let \ea \fontswitch \the font \def \currvar{\fontswitch}
\endgroup

\protected \def \fontdef #1#2{%
edef \xfamv{\csstring#1}
let \ttfamv = \xfamv \ea \endgroup \the font \def \currvar{#1}
}
\protected \edef \tmpa {famdepend#1\famv \csstring#1}
\ifx #1 \tt
\protected \def \tt{\def \xfamv{tt} \ea \the font \def \currvar{\tt}}
let \tt = \tt
\else \setnewmeaning #1 = \tmpa \famvardef
\fi
}
\public \fontdef ;

The \famvardef \xxx \{\data\} does, roughly speaking:
def \xxx \{\data\ea\the font \def \currvar{\xxx}\}

but the macro \xxx is declared as family-dependent. It is analogically as in \moddef. The \xxx is defined as \famdepend\xxx f:\\currfamily:xxx} and \f:\{\currfamily}:xxx} is defined as mentioned.

\famvardef\tt behaves somewhat differently: it defines internal version \_tt (it is used in \_ttfont and \_urlfont) and set \tt to the same meaning.

The \fontfam \{\Font Family\} does:
- Convert its parameter to lower case and without spaces, e.g. \fontfamily.
- If the file \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.

\protected\def \fontfamily [#1]{% \lowercase{\edef\famname{\removespaces \expanded{#1} {}}} \isfile \f\famname.opm \iftrue \edef\famfile{f\famname} \opinput \f\famname.opm \else \tryloadfamslocal \edef\famfile{\trycs{fam:\famname}{}} \ifname \fams:f\famname endname \edef\famfile{f\famname} \famsfams \else \listfamnames \fi \else \opinput \famfile.opm \fi \empty \fi \def \tryloadfamslocal{% \isfile \fams-local.opm \iftrue \opinput \fams-local.opm \famfrom={} \famsrc={} \fi \let \tryloadfamslocal=\relax % need not to load fams-local.opm twice \def \listfamnames {% \wterm{===== List of font families ======} \begingroup \let \famtext=\wterm \def \famins[#1][#2][#3][#4]{% \wterm{ \space \noexpand\fontfam [##1] -- ##2} \let \famaliasA=\famalias \opinput \fams-local-opm \iftrue \fams=\fams-local-opm \famfrom={} \famsrc={} \fi \let \tryloadfamslocal=\relax % need not to load fams-local.opm twice \endgroup \def \famaliasA{\wterm {FONT-SUB: \famfile \space \famalias \space \famins}} \def \famaliasA{\wterm{\message{\famins}}} \def \famins[#1][\message[#1]]{\famins} \endgroup} \public \fontfam ;

\fontfamsub \{[\Family] \{(byFamily)\} \} declares automatic substitution of \Family by \byFamily which is done when \Family is not installed. I.e. if there is no \f\Family.opm file or there is no regular font of the family installed. \famsfams is internal macro used in \fontfam and \famdecl macros. It consumes the rest of the macro, runs \nospacefuturelet in order to do \endinput to the current f-file and runs \fontfam again. The table of such substitutions are saved in the macros \fams{\family-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\}\} \{\{mod-plus-vars\}\} \{\{mod-plus-vars\}\} does nothing or prints the text on the terminal. The \{mod-plus-vars\} data are used when printing the font catalog.

\_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 is printed in the font catalog.

\_famsrc declares the source, where is the font family from (used in fams-ini.opm and if the font isn’t found when the fonts catalog is printed).

\newtoks \_famfrom \_newtoks \_famsrc
\_input fams-ini.opm
\_let \_famfile=\_undefined
\_famfrom={} \_famsrc={}

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, \catalogexclude and \catalognextfam of the catalog are declared here.

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

- \_defaultfontfeatures – used for each font,
- \_fcolor – features added by \setfontcolor (this is obsolete)
- \_ffletterspace – features added by \setletterspace,
- \_ffwordspace – features added by \setwordspace.

The macros \_ffadded, \_ffcolor, \_ffletterspace, \_ffwordspace are empty by default.

The \_setff \{\{features\}\} adds next font features to \_ffadded. Usage \_setff{} resets empty set of all \_ffadded features.

The \_famtext \{\{Family Name\}\} \{\{comments\}\} \{\{file-name\}\} \{\{mod-plus-vars\}\} does
The \setletterspace command is based on the special font features provided by the \luaotfload package. The \setwordspace command recalculates the \fontdimen\texttt{2,3,4} of the font using the \setwsp command, which is used by the \_fontselA macro. It activates a dummy font feature \texttt{+Ws} too in order the font is reloded by the \font primitive (with independent \fontdimen registers). If the \setwordspace is used again to the same font then we need to reset \fontdimen registers first. It is done by \_sws:⟨\texttt{fontname}⟩ macro which keeps the original values of the \fontdimens.

\setfontcolor is kept here only for backward compatibility but not recommended. Use real color switches and the \transparency instead.

\begin{verbatim}
\_def \_setfontcolor #1{%
  \edef \_tmp{\_calculatefontcolor{#1}}%\_ifx \_tmp \_empty \_def \_ffcolor{}\_else \edef \_ffcolor{color=\_tmp;}\_fi
}\_def \_setletterspace #1{%
  \_setfontcolor \_if\#1\_def \_fletterspace{}\_else \def \_fletterspace{letterspace=#1;}\_fi
}\_def \_setwordspace #1{%
  \_setfontcolor \_if\#1\_def \_fwordspace{}\_def \_ffwordspace{}%\_else \def \_fwordspace{\_setwordspaceA#1/}\def \_ffwordspace{+Ws;}\_fi
}\_def \_setwordspaceA #1{\_ifx/#1\_ea \_setwordspaceB \_else \_afterfi \_setwordspaceC#1}\_fi
\_def \_calculatefontcolor#1{\_trycs{\_fc:#1}{#1}} % you can define more smart macro ...
\_sdef{\_fc:red}{FF0000FF} \_sdef{\_fc:green}{00FF00FF} \_sdef{\_fc:blue}{0000FFFF}\_sdef{\_fc:yellow}{FFFF00FF} \_sdef{\_fc:cyan}{00FFFFFF} \_sdef{\_fc:magenta}{FF00FFFF} \_sdef{\_fc:white}{FFFFFFFF} \_sdef{\_fc:grey}{00000080} \_sdef{\_fc:lgrey}{00000025} \_sdef{\_fc:black}{}
\_public \setfontcolor \setletterspace \setwordspace;
\end{verbatim}

\_regoptsizes \texttt{(internal-template)} \texttt{(left-output)}?\texttt{(right-output)} \texttt{(resizing-data)} prepares data for using by the \_optname \texttt{(internal-template)} macro. The data are saved to the \texttt{\_oz:(internal-template)} macro. When the \_optname is expanded then the data are scanned by the macro \_optnameA \texttt{(left-output)}?\texttt{(right-output)} \texttt{(mid-output)} \texttt{(<size>)} in the loop.

\_optfontalias \{\texttt{(template A)}\} \{\texttt{(template B)}\} is defined as \texttt{\_let\_oz:(templateA)=\_oz:(templateB)}.

\begin{verbatim}
\_def \_regoptsizes #1 #2?#3 #4*{\_sdef{\_oz:#1}{#2?#3 #4*}}
\_def \_optname #1{\_ifcsname \_oz:#1\_endcsname \_optnameA \csname \_oz:#1\_endcsname \_else \_failedoptname{#1}\_fi}
\_def \_failedoptname #1{optname-fails:(#1)}\_def \_optnameA #1?#2 #3 <#4 {\_ifx*#4#1#3#2\_optnameC\_else \_ifdim \_optsize<#4pt #1#3#2\_optnameC\_else \_afterfifi \_optnameA #1?#2\_fi\_fi}
\_def \_optnameC #1* {\_fi\_fi}
\_def \_afterfifi #1\_fi\_fi\_fi{\_fi\_fi #1}
\_def \_optfontalias #1#2{\_slet{\_oz:#1}{\_oz:#2}}
\_setfontsize {at10pt} % default font size
\end{verbatim}

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 OpTEX says: set three sizes by the macro \texttt{\setmathsizes \{[\text-size]/[\script-size]/[\scriptscript-size]\}} and then load all math fonts in given sizes by \texttt{\normalmath} or \texttt{\boldmath} macros. For example

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

We have two math macros \texttt{\normalmath} 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 \_script \{\_fam6 \} % more extensive script than \cal
\_public \mit \cal \bbchar \frak \script ;

The family is saved as \_fam \langle \number \rangle.

\_def \_loadmathfamily \{\_afterassignment\_loadmathfamilyA \_chardef\_mfam\}
\_def \_loadmathfamilyA \#1 \{\_mfactor \edef\_optsizesave\{\_the\_optsize\}%
\_optsize=\_sizemtext \_font\_mF \_optfn{#1} at\_optsize \_textfont\_mfam=\_mF
\_optsize=\_sizemscript \_font\_mF \_optfn{#1} at\_optsize \_scriptfont\_mfam=\_mF
\_optsize=\_sizemsscript \_font\_mF \_optfn{#1} at\_optsize \_scriptscriptfont\_mfam=\_mF
\_optsize=\_optsizesave
\__loadmathfamily \langle \number \rangle \langle \font \rangle loads one math family, i.e. the triple of fonts in the text size, script size and script-script size. The \langle \font \rangle is \langle \font-id \rangle used in the \_regtfm parameter or the real TFM name.

The optical sizes of Computer Modern fonts, AMS, and other fonts are declared here.

\_loadmathfamily \langle \number \rangle \langle \font \rangle loads one math family, i.e. the triple of fonts in the text size, script size and script-script size. The \langle \font \rangle is \langle \font-id \rangle used in the \_regtfm parameter or the real TFM name. The family is saved as \_fam \langle \number \rangle.

\_setmathfamily \langle \number \rangle \langle \font-switch \rangle loads one math family like \_loadmathfamily does it. But the second parameter is a \langle \font-switch \rangle declared previously by the \font primitive.

The \langle \number \rangle is saved by \_loadmathfamily, \_setmathfamily to the \_mfam. The font family is loaded at \_sizemtext, \_sizemscript and \_sizemsscript sizes. These sizes are set by the \_setmathsizes \{[text-size]/[script-size]/[scriptscript-size]\} macro. These parameters are given in the ptmunit unit, it is set to \ptunit and it is set to 1pt by default.

\_mfactor sets scaling factor for given math fonts family related to text font size. It does the setting \_ptmunit=\langle \factor \rangle \ptunit where the \langle \factor \rangle is defined by \_def\_mfactor: \langle \factor \rangle\}. For example, you can set \_def\_mfactor: 1\{0.95\} if you found that this scaling of math family 1 gives better visual compatibility with used text fonts. If not declared then scaling factor is 1.

\_def\_loadmathfamilyA \{\_afterassignment\_loadmathfamilyA \_chardef\_mfam\}
\_def\_loadmathfamilyA \#1 \{\_mfactor \edef\_optsizesave\{\_the\_optsize\}%
\_optsize=\_sizemtext \_font\_mF \_optfn{#1} at\_optsize \_textfont\_mfam=\_mF
\_optsize=\_sizemscript \_font\_mF \_optfn{#1} at\_optsize \_scriptfont\_mfam=\_mF
\_optsize=\_sizemsscript \_font\_mF \_optfn{#1} at\_optsize \_scriptscriptfont\_mfam=\_mF
\_optsize=\_optsizesave

\_loadmathfamily \langle \number \rangle \langle \font \rangle loads one math family, i.e. the triple of fonts in the text size, script size and script-script size. The \langle \font \rangle is \langle \font-id \rangle used in the \_regtfm parameter or the real TFM name. The family is saved as \_fam \langle \number \rangle.

\_setmathfamily \langle \number \rangle \langle \font-switch \rangle loads one math family like \_loadmathfamily does it. But the second parameter is a \langle \font-switch \rangle declared previously by the \font primitive.

The \langle \number \rangle is saved by \_loadmathfamily, \_setmathfamily to the \_mfam. The font family is loaded at \_sizemtext, \_sizemscript and \_sizemsscript sizes. These sizes are set by the \_setmathsizes \{[text-size]/[script-size]/[scriptscript-size]\} macro. These parameters are given in the ptmunit unit, it is set to \ptunit and it is set to 1pt by default.

\_mfactor sets scaling factor for given math fonts family related to text font size. It does the setting \_ptmunit=\langle \factor \rangle \ptunit where the \langle \factor \rangle is defined by \_def\_mfactor: \langle \factor \rangle\}. For example, you can set \_def\_mfactor: 1\{0.95\} if you found that this scaling of math family 1 gives better visual compatibility with used text fonts. If not declared then scaling factor is 1.
The \_setmathdimens macro is used in \normalmath or \boldmath macros. It makes math dimensions dependent on the font size (plain \TeX sets them only for 10 pt typesetting). The \skewchar of some math families are set here too.

\_setmathparams{\Umathspaceafterscript} is used instead of \scriptspace setting because Lua\TeX ignores \scriptspace in most cases. There is small difference from classical \TeX: we set “scaled” \Umathspaceafterscript dependent on textstyle, scriptstyle, etc. sizes. The \_scriptspacefactor is set to 0.05 which gives the same result as Plain \TeX \scriptspace=0.5 pt at 10 pt font size.

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

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

There is a problem: The \textit{x_n} is tokenized as \textit{x} \_ \textit{n} and it works without problems. But \texttt{\int_a^b} is tokenized as \texttt{\int_a \_ \_ 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 \rangle \_ \langle one-letter \rangle} (where \texttt{\langle word \rangle} is a sequence of letters) because such control sequences are inaccessible: preprocessor rewrites it.

The \texttt{\mathsbt} macro activates the rewriting rule \texttt{\langle word \rangle \_ \langle nonletter \rangle} to \texttt{\langle word \rangle \_ \langle nonletter \rangle} and \texttt{\langle word \rangle \_ \langle letter \rangle \langle nonletter \rangle} to \texttt{\langle word \rangle \_ \langle letter \rangle \langle nonletter \rangle} at input processor level. The \texttt{\mathsbof} deactivates it. You can ask by \_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.
All control sequences declared by \texttt{\mathchardef} are supposed (by default) only for public usage. It means that they are declared without \texttt{_} prefix. If such sequences are used in internal OpTEX macro then their internal prefixed form is declared using \texttt{_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 plain\TeX, but they are \texttt{protected} in OpTEX.

\texttt{\protected\def\log{\_mathop{\_rm log}\_nolimits}}
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.

\texttt{\sp} is an alternative for \^{}. The \texttt{\sb} alternative for _ was defined at line 27 of the file \texttt{math-macros.opm}.

Active \texttt{\prime} character is defined here.
\kern\nulldelimiterspace\right.)}}
\def\scalebigcoefs#1{% 
\ifcase #1 0 \or
.6 \or .72 \or .9 \or 1.2 \or 1.5 \else 0 \fi
}
\protected\def\big#1{% \scalebig{#1}1}
\protected\def\bbig#1{% \scalebig{#1}2}
\protected\def\Big#1{% \scalebig{#1}3}
\protected\def\bigg#1{% \scalebig{#1}4}
\protected\def\Bigg#1{% \scalebig{#1}5}
\public \big \bbig \Big \bigg \Bigg ;
\protected\def\bigl{% \mathopen\big}
\protected\def\bigm{% \mathrel\big}
\protected\def\bigr{% \mathclose\big}
\protected\def\bbigl{% \mathopen\bbig}
\protected\def\bbigm{% \mathrel\bbig}
\protected\def\bbigr{% \mathclose\bbig}
\protected\def\Bigl{% \mathopen\Big}
\protected\def\Bigm{% \mathrel\Big}
\protected\def\Bigr{% \mathclose\Big}
\protected\def\biggl{% \mathopen\bigg}
\protected\def\biggm{% \mathrel\bigg}
\protected\def\biggr{% \mathclose\bigg}
\protected\def\Biggl{% \mathopen\Bigg}
\protected\def\Biggm{% \mathrel\Bigg}
\protected\def\Biggr{% \mathclose\Bigg}
\public \bigl \bigm \bigr \bbigl \bbigm \bbigr \Bigl \Bigm \Bigr \biggl \biggm \biggr \Biggl \Biggm \Biggr ;
\ldots \cdots \vdots \ddots from plain TeX
\protected\def\ldots{% \mathinner{\ldotp \ldotp \ldotp}}
\protected\def\cdots{% \mathinner{\cdotp \cdotp \cdotp}}
\protected\def\vdots{% \vbox{\baselineskip=.4em \lineskiplimit=\zolineskip
\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}}
\public \ldots \cdots \vdots \ddots ;
\ldots , \cdots , \vdots , \ddots from plain TeX
\protected\def\ldotp{% \mathinner{\ldotp \ldotp \ldotp}}
\protected\def\cdotp{% \mathinner{\cdotp \cdotp \cdotp}}
\dots inspired by plain \TeX

\protected\def\adots{\mathinner{%
  \mkern1mu\raise1em\hbox{.}\mkern2mu\raise.4em\hbox{.}\mkern2mu\raise.7em\vbox{\kern.7em\hbox{.}}\mkern1mu}}

\public \adots ;

\protected\def\acute{\mathaccent"7013}
\protected\def\grave{\mathaccent"7012}
\protected\def\ddot{\mathaccent"707F}
\protected\def\tilde{\mathaccent"707E}
\protected\def\bar{\mathaccent"7016}
\protected\def\breve{\mathaccent"7015}
\protected\def\check{\mathaccent"7014}
\protected\def\hat{\mathaccent"705E}
\protected\def\vec{\mathaccent"017E}
\protected\def\dot{\mathaccent"705F}
\protected\def\widetilde{\mathaccent"0365}
\protected\def\widehat{\mathaccent"0362}

\math, \skew, \overrightarrow, \overleftarrow, \overbrace, \underbrace macros. The last four are redefined when Unicode math is loaded.

\protected\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{\mathialign{##\crcr\rightarrowfill\crcr
\noalign{\kern-.1em\nointerlineskip}$\hfil\displaystyle{#1}\hfil$\crcr}}}
\protected\def\overleftarrow #1{\vbox{\mathialign{##\crcr\leftarrowfill\crcr
\noalign{\kern-.1em\nointerlineskip}$\hfil\displaystyle{#1}\hfil$\crcr}}}
\protected\def\overbrace #1{\mathop{\vbox{\mathialign{##\noalign{\kern.3em}$\downbracefill\crcr
\noalign{\kern.3em\nointerlineskip}$\hfil\displaystyle{#1}\hfil$\crcr}}\limits}
\protected\def\underbrace #1{\mathop{\vtop{\mathialign{##\noalign{\kern.3em}$\upbracefill\crcr
\noalign{\kern.3em}}}$\hfil\displaystyle{#1}\hfil$\crcr\limits}}

\public \overrightarrow \overleftarrow \overbrace \underbrace \skew ;

Macros based on \delimiter, \*witdelims and \radical primitives.
\protected\def\choose{\atopwithdelims()}
\protected\def\brack{\atopwithdelims[{}]}\brack{\atopwithdelims\lbrace\rbrace}
\protected\def\sqrt{\radical"270370 } \sqrt ;\mathpalette,
\vphantom, \hphantom, \phantom, \mathstrut, and \smash macros from plain \TeX.

\protected\def\mathpalette#1#2{\mathchoice{#1\displaystyle{#2}}%\mathchoice{#1\textstyle{#2}}{#1\scriptstyle{#2}}{#1\scriptscriptstyle{#2}}}
\newbox\rootbox\protected\def\root#1\of{\setbox\rootbox=\hbox{$\math\scriptscriptstyle{#1}$}\mathpalette\rootA}
\def\rootA#1#2{\setbox0=\hbox{$\math#1\sqrt{#2}$}\dimen0=\ht0 \advance\dimen0by-\dp0 \mkern5mu\raise.6\dimen0\copy\rootbox \mkern-10mu\box0}
\newifi\ifvp \newifi\ifhp\protected\def\vphantom{\vptrue\hpfalse\phant}
\protected\def\hphantom{\vpfalse\hptrue\phant}\protected\def\phantom{\vptrue\hptrue\phant}
\def\phant{\ifmmode\def\next{\mathpalette\mathphant}\else\let\next=\makenphant\fi\next}
\def\makenphant#1{\setbox0=\hbox{#1}\finphant}
\def\mathphant#1#2{\setbox0=\hbox{$\math#1{#2}$}\finphant}
\def\finphant{\setbox2=\null\ifvp\ht2=\ht0\dp2=\dp0\fi\ifhp\wd2=\wd0\fi\hbox{\box2}}\mathstrut, and \smash 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\icancel\in}}\def\icancel#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$#1\leftharpoonup$}}}}\protected\def\buildrel#1\over#2{\mathrel{\mathop{\kern\z@ #2}\limits^{#1}}}
\protected\def\doteq{\buildrel\textstyle.\over=}\in \sim ; \public \cong \notin \rightleftharpoons \buildrel \doteq \; \bmod \and \pmod from \TeX.\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.
The \texttt{cases} and \texttt{bordermatrix} macros are almost identical as in plain \TeX. You can simply re-define \texttt{bordermatrix} with other delimiters using the common \texttt{\begin{bordermatrix}withdelims} macro.

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

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

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

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

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

95
\openup, \leqalignno and \leqalignno macros are copied from Plain \TeX unchanged.

These macros are inspired by ams-math.tex file.

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

\not< becomes \_nless
\not> becomes \_ngtr
if \_notXXX is defined, \not\XXX becomes \_notXXX;
if \_nXXX is defined, \not\XXX becomes \_nXXX;
otherwise, \notXXX is done in the usual way.
\edef\tmpn{\csstring#1}\%
\ifcsname _not\tmpn\endcsname \csname _not\tmpn\endcsname
\else \mathrel{\notchar#1}\%
\fi \fi \fi \fi}
\private
\leq \ngeq \less \gtr \prec \succ \leqslant \ngeqslant \preceq \succeq \leqq \ngeqq \sim \cong \subseteqq \supseteqq \subseteq \supseteq \parallel \mid \shortmid \shortparallel \vdash \Vdash \vDash \VDash \trianglerighteq \trianglelefteq \triangleright \leftarrow \rightarrow \Leftarrow \Rightarrow \Leftrightarrow \leftrightarrow \exists ;
\public \not ;
\mathstyles{\langle \text{math list} \rangle} behaves like {\langle \text{math list} \rangle}, but you can use the following commands in the \text{math list}:

- \texttt{currstyle} which expands to \texttt{\displaystyle}, \texttt{\textstyle}, \texttt{\scriptstyle} or \texttt{\scriptscriptstyle} depending on the current math style when \texttt{\mathstyles} was opened.
- \texttt{dobystyle}(⟨D⟩,⟨T⟩,⟨S⟩,⟨SS⟩) is expandable macro. It expands to ⟨D⟩, ⟨T⟩, ⟨S⟩ or ⟨SS⟩ depending on the current math style when \texttt{\mathstyles} was opened.
- The value of the \texttt{\stylenum} is 0, 1, 2 or 3 depending on the current math style when \texttt{\mathstyles} was opened.

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

The \texttt{\cramped} macro sets the cramped variant of the current style. Note that \texttt{\currstyle} initializes non-cramped variants. The example \texttt{\mathframe} above should be: \texttt{\def\mathframe#1{\mathstyles{\frame{$\currstyle\cramped #1$}}}}.

Second note: \texttt{\cramped} macro reads the current math style from the \texttt{\mathstyle} LuaTeX primitive, so it does not work in numerators of generalized fractions but you can use it before the fraction is opened: \texttt{\cramped {x^2\over y^2}}.

\texttt{\setmathstyle} saves current math style (including its cramped/normal subversion) and \texttt{\usemathstyle} restores the saved math style. These macros are based on the LuaTeX’s \texttt{\mathstyle} primitive, i.e. they don’t work in generalized fractions.

Usage: \texttt{\def\mathclap #1{{\setmathstyle \hbox to0pt{\hss$\usemathstyle#1$\hss}}}}.

The \texttt{\mathbox{⟨text⟩}} macro is copied from OPmac trick 078. It behaves like \texttt{\hbox{⟨text⟩}} but the ⟨text⟩ is scaled to a smaller size if it is used in \texttt{\scriptstyle} or \texttt{\scriptscriptstyle} style. The \texttt{\textmff} and \texttt{\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 \texttt{\mathbox} in non-Unicode math respects optical sizes using different principle.
2.16 Unicode-math fonts

The \loadmath \langle optional-factor \rangle \{\langle Unicode-math font \rangle\} macro loads the given math font 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.

The \langle optional-factor \rangle is scaling factor of loaded font with respect to the size of the text font. It can be used if the used text font and loaded math font have incompatible ex height. If missing then the scaling factor is 1.

\loadmath \{\langle bold-font \rangle\} \to \{\langle normal-font \rangle\} loads bold variant only if \langle normal-font \rangle was successfully 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 main math font is loaded by \loadmath (typically indirectly using \fontfam) and you can load more additional math fonts by \addUmathfont:

\addUmathfont \famname \{[\langle normal-font \rangle\} \{ffeatures\} \{[\langle bold-font \rangle\} \{ffeatures\} \{[\langle factor \rangle\}

The \famname is a control sequence declared by \addUmathfont for later use. It gets math family number. The \langle factor \rangle is decimal number for size corrections in view of the main math font. If it is empty then \langle factor \rangle=1. If \langle bold-font \rangle is empty, the “faked bold” derived from \langle normal-font \rangle is used. Example:

\fontfam[\lm] \% does \loadmath{\[latinmodern-math\]}
\addUmathfont \famname \{[XITSMath-Regular]\}\{[XITSMath-Bold]\}\{\}

declares latinmodern-math as main math font (its bold variant is “faked bold”). The additional math font family \famname is declared in the example. It uses XITSMath-Regular for normal printing and XITSMath-Bold for bold printing.

All characters used in math formula are printed from main math font by default. But you can redeclare characters for printing from additional font by \mathchars \famname \{[\langle list of sequences \rangle\]. For example:

\mathchars \famname \{\stareq \triangleq \veeq \wedgeq\}

sets the characters \stareq, \triangleq, \veeq, \wedgeq from the \famname additional font. The \{\langle list of sequences \rangle\} can include control sequences from the unicode-table.tex, but no math accents. These control sequences can be printed by \input print-unimath.opm.

The \mathchars macro keeps the class and slot of declared math objects and re-declares only family of them. It is applied to all control sequences given in the parameter. The relevant math codes are re-declared.

Use \addto\selector\fam\famname if you want to print whole math alphabet from an additional math font. For example \addto\cal\fam\famname declares all \cal characters from the \famname font loaded by \addUmathfont.

The \mathcodes macro provides comfortable settings of math codes of math objects. Its syntax is \mathcodes \{family\} \{[\langle list-of-pairs \rangle\}. Each pair in the \langle list-of-pairs \rangle is \{[class-number\} \{character\} (separated by optional space) or \{[class-number\} \{[list-of-characters\}. The \{list-of-characters\} includes declared characters or \Urange \langle from \rangle-\langle to \rangle which is equal to the list of characters beginning \langle from \rangle and ending \langle to \rangle, for example \Urange a-d is equal to abcd. The characters can be given directly or by the math sequences like \times, \doteq too.

The \mathcodes macro declares mathcode of given characters internally by
The `\mathcode` macro sets math codes of given Unicode characters. The relevant control sequence from `unicode-table.tex` changes its behavior too. For example, if you change math code of × then the `\times` control sequence will behave like new declared ×.

### 2.16.1 Unicode-math macros preloaded in the format

```
\newif\ifmathloading
\def\noloadmath{\mathloadingfalse}
\def\doloadmath{\mathloadingtrue}
```

---

The `\loadmath` macro sets math codes of given Unicode characters. The relevant control sequence from `unicode-table.tex` changes its behavior too. For example, if you change math code of × then the `\times` control sequence will behave like new declared ×.

---

The Unicode version of the `\normalmath` and `\boldmath` macros are defined here as `\_normalunimath` and `\_boldunimath` macros. They are using `\_setunimathdimens` in a similar sense as `\_setmatdimens`. You can combine more fonts if you register them to another math families (5, 6, 7, etc.) in the `\normalmath` macro.

The default value of `\_normalunimath` shows a combination of base Unicode-math font at family 1 with 8bit Math font at family 4. See definition of `\script` macro where `\fam4` is used.
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{%
  \loadmathfamily1 {{xitsmath-bold}} \% Base font
  \loadmathfamily4 \textsf \% script
  \loadmathfamily5 {{xitsmath-regular}} \%
  \def\|{\Udelimiter 0 5 "02016 }% % norm delimiter from family 5
}\setmathdimens{}
```

`\loadmathfamily (number) {{(font)}}{(font features)} loads the given Unicode-math fonts in three sizes using single `(font)` with different `mathsize=1,2,3` font features. The math font family is set with `\_mfam` and the size-dependent features `ssty=1` if script size is asked or `ssty=2` if scriptscriptsize is asked.

`\mparams` can insert additional font features depending on the current `\_mfam`.

The `\textmff`, `\scriptmff` and `\sscriptmff` are font features for text, script and scriptscript sizes respectively. They are locally re-defined in `\mathbox` macro.
\neufam. We use \theadto here because we want to read the main family 1 as last one (for definitive setting of math parameters).

The math characters can be given directly (by their Unicode) or by a macro like \doteq, \times, etc. These macros simply expand to the math character with its Unicode. And this math character has its \Umathcode given by \langle class \rangle, \langle family \rangle, \langle slot-number \rangle. Sometimes, we may want to get these quantities from the given Unicode math character by our macros. It is possible by \themathcodeclass ⟨math-char⟩, \themathcodefam ⟨math-char⟩ and \themathcodechar ⟨math-char⟩ macros. The parameter ⟨math-char⟩ is a math character or it is a macro like \doteq, \times. Moreover, \thedelcodefam ⟨math-char⟩ and \thedelcodechar ⟨math-char⟩ return delcode quantities of given math character.

\mathchars (fam) \langle list of sequences \rangle saves \langle fam \rangle to \_mafam and runs for each sequence from the \langle list of sequences \rangle the relevant code settings using \Umathcode primitive. In case of \\int-like operators the \langle math class \rangle=8 and we only re-declare \_int:⟨int-character⟩ as an operator with the new \_mafam. Note that the used primitives have the syntax:

\Umathchardef \langle sequence \rangle = \langle math class \rangle \langle math family \rangle \langle slot number \rangle
\Umathcode \langle code \rangle = \langle math class \rangle \langle math family \rangle \langle slot number \rangle
\Udelcode \langle code \rangle = \langle math family \rangle \langle slot number \rangle

\mathcodes (fam) \langle list of pairs \rangle sets mathcodes of given characters with explicit \langle class \ranglees. Each pair can be \langle class \rangle \langle list of chars \rangle and \langle list of chars \rangle can include \\Urange \langle from \rangle – \langle to \rangle. This is reason why we apply \\expanded to the \langle list of chars \rangle before reading it by \foreach: the \\expanded is expandable and expands to the relevant list of characters.
2.16.2 Macros and codes set when \loadmath is processed firstly

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

Unicodemath font includes all typical math alphabets together, user needs no 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.

\umathrange \{\langle from\rangle \rightarrow \langle to\rangle \}\{\langle family\rangle \}\{\langle first\rangle\} sets \mathcodes of the characters in the interval \langle from\rangle \rightarrow \langle to\rangle to \langle first\rangle+1, \langle first\rangle+2 etc., but \umathcharholes are parts of the Unicode table not designed for math alphabets, 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 \langle from\rangle \rightarrow \langle to\rangle clause includes characters like A–Z. Note that the \umathrange sets the \classfam macro as \langle class\rangle \langle family\rangle for later use.

\umathrange \{\langle from\rangle \rightarrow \langle to\rangle \}\{\langle class\rangle \}\{\langle family\rangle \}\{\langle first\rangle\} sets \mathcodes of the characters in the interval \langle from\rangle \rightarrow \langle to\rangle to \langle first\rangle+1, \langle first\rangle+2 etc., but \umathcharholes are parts of the Unicode table not designed for math alphabets, 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 \langle from\rangle \rightarrow \langle to\rangle clause includes characters like A–Z. Note that the \umathrange sets the \classfam macro as \langle class\rangle \langle family\rangle for later use.

A few math characters have very specific Unicode and must be handled individually. We can run the \umathrangespec macro just after the \umathrange. The \umathnumA must be set to the first destination code. The \umathrangespec applies to each character from the \{list of characters\} this: \mathcode(\char-code)=\classfam(\umathnumA) and increments \umathnumB. If \umathnumB=0 then it applies \mathcode(\char-code)=\classfam(\char-code) \classfam. The \classfam and \umathnumB were typically set by previous call of the \umathrange macro.

The math alphabets are set by \rmvariables, \bfvariables, \itvariables, \bivariables, \calvariables, \bcalvariables, \frakvariables, \bfrakvariables, \bbvariables, \sansvariables, \bsansvariables, \bsansgreekvariables, \ttvariables, \itgreek, \rmgreek, \bfgreek, \bigreek, \bsangreek, \bsangreek, \itGreek, \rmGreek, \bfGreek, \bigreek, \bsangreek, \bsangreek, \rmdigits, \bmdigits, \bbdigits, \sansdigits, \bsansdigits, \ttdigits. They are declared using the \umathrange(\langle range\rangle)(\langle class\rangle)(\langle family\rangle)(\langle starting-code\rangle) macro.
The control sequences for \( \alpha \), \( \beta \), etc. are redefined here. The \( \alpha \) will expand to the character with Unicode "03B1", this is a normal character \( \alpha \). You can type it directly in your editor if you know how to do this. These sequences are declared by \_greekdef\( \langle \)list of sequences\( \rangle \relax \).
The \texttt{\_umathrangeGREEK} \texttt{(class)/(family)/(first)} and \texttt{\_umathrangegreek} \texttt{(class)/(family)/(first)} macros for setting math codes of Greek characters are defined here. They use \texttt{\_umathrange} for general codes but the exceptions must be handled by the \texttt{\_umathrangespec} macro. The exceptions are seven Greek characters: \(\varepsilon, \vartheta, \varkappa, \varphi, \varrho, \varpi, \nabla\). The first six of these characters should behave as lowercase Greek letters and the last one \texttt{\nabla} is uppercase Greek letter.

\begin{verbatim}
\_protected\_def\_mabf {\_inmath{\_bfvariables\_bfGreek\_bfdigits}}
\_protected\_def\_mabi {\_inmath{\_bivariables\_bigreek\_bfGreek\_bfdigits}}
\_inmath {\_cmds} \texttt{applies} \{\_cmds\} \texttt{only in math mode.}
\end{verbatim}
Each Unicode slot carries information about math type. This is saved in the file `MathClass-15.txt` which is copied to `mathclass.opm`. The file has the following format:

```
70 002E;P
71 002F;B
72 0030..0039;N
73 003A;P
74 003B;P
75 003C;R
76 003D;R
77 003E;R
78 003F;P
79 0040;N
80 0041..005A;A
81 005B;O
82 005C;B
83 005D;C
84 005E;N
85 005F;N
```

We have to read this information and convert it to the `\Umathcode`s.

```
242 \begingroup % \input mathclass.opm (which is a copy of MathClass.txt):
243 \long\def\p#1#2\{\ifx^#2^\else
244 \edef\tmp{\csname _c:#2\endcsname}\if\relax\tmp\else \pA#1\..\end#2\fi
245 \ea\p \fi }
246 \def\pA#1..#2..#3\end#4{\
247 \ifx\relax#2\relax \pset{#1}{#4}\else \fornum{#1..}#2\do{\pset{##1}{#4}}\fi
248 }
249 \def\pset#1#2{\Umathcode#1=\tmp\space1#1\relax
250 \if#2O\Udelcode#1=1#1\relax\fi
251 \if#2C\Udelcode#1=1#1\relax\fi
252 \if#2F\Udelcode#1=1#1\relax\fi
253 }
254 \catcode`#=14 \everyeof={;{} }
255 \globaldefs=1 \ea\p \input mathclass.opm
256 \endgroup
```

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

```
36 \UnicodeMathSymbol{000B1}{\pm }{\mathbin}{plus-or-minus sign}%
37 \UnicodeMathSymbol{000B6}{\mathparagraph }{\mathord}{paragraph symbol}%
38 \UnicodeMathSymbol{000B7}{\cdotp }{\mathbin}{/centerdot b: middle dot}%
39 \UnicodeMathSymbol{D7}{\times }{\mathbin}{multiply sign}%
40 \UnicodeMathSymbol{000F7}{\matheth }{\mathalpha}{eth}%
41 \UnicodeMathSymbol{000F7}{\div }{\mathbin}{divide sign}%
```

We have to read this information and set given control sequences as macros which expand to the given Unicode character. This solution enables to use such control sequences in PDF outlines where they expand to the appropriate Unicode character. We don’t use `\mathchardef`, we set the mathcodes (class, family, slot) only at single place: for Unicode math characters. For example for we define `\times`:

```
\def\times{\^d7} \Umathcode "D7 = 2 1 "D7
```

Because math codes of Greek upright letters vary depending on `\_itgreek`, `\_bfgreek`, etc. macros, we need to keep the access directly to these characters. We define `\mupalpha`, `\mupbeta`, ..., `\mupomega` macros as a code from PUA (Private Use Area) of Unicode table and set mathcode of these codes to the real upright alpha, beta, ..., omega.

```
\begingroup % \input unimath-table.opm (it is a copy of unicode-math-table.tex):
\umathtabnum=7800 % pointer to the Private User Area
\def\UnicodeMathSymbolSymbol#1#2#3#4{\edef#1\csname _Uchar#2\endcsname}
\edef\Uchar{\char{}\Udelcode{000077}}\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\times\\time
```

105
The macro \int expands to an \langle int-character \rangle. We save the \mathcode of the \langle int-character \rangle using \Umathchardef and declare \langle int-character \rangle as math-active and define it as \langle int-character \rangle \_nolimits. Moreover, we define \intop as \langle int-character \rangle (it is the integral with limits like in plain \TeX). We do this with other int-like operators listed below too.

Many special characters must be declared with care...

\global \Udelcode`<=1 "027E8 \% these characters have different meaning
\global \Udelcode`>=1 "027E9 \% as normal and as delimiter
\mit \% default math alphabets setting
% hyphen character is transformed to minus:
\Umathcode `\^ = 2 1 "2212
% mathclass defines : as Punct, plain.tex as Rel, we keep mathclass,
% i.e. there is difference from plain.tex, you can use $f:A\to B$.
% mathclass defines ! as Ord, plain.tex as Close
\Umathcode `! = 5 1 `! \% keep plain.tex declaration
% mathclass defines ? as Punct, plain.tex as Close
\Umathcode `? = 5 1 `? \% keep plain.tex declaration
\Umathcode `* = 2 1 "02217 \% equivalent to \ast, like in plain \TeX
\Umathcode "03A2 = 7 1 "03F4 \% \varTheta
\Umathcode `© = 0 1 `© \% usage $\copyright$ can be seen in old documents
\protected\def \_sqrt {\_Uradical 1 "0221A }
\protected\def \_cuberoot {\_Uradical 1 "0221B }
\protected\def \_fourthroot {\_Uradical 1 "0221C }
\public \sqrt \cuberoot \fourthroot ;
\protected\def \_overbrace #1{\_mathop {\_Umathaccent 7 1 "023DE{#1}}\_limits}
\protected\def \_underbrace #1{\_mathop {\_Umathaccent bottom 7 1 "023DF{#1}}\_limits}
\protected\def \_overparen #1{\_mathop {\_Umathaccent 7 1 "023DC{#1}}\_limits}
\protected\def \_underparen #1{\_mathop {\_Umathaccent bottom 7 1 "023DD{#1}}\_limits}
\protected\def \_overbracket #1{\_mathop {\_Umathaccent 7 1 "023B4{#1}}\_limits}
\protected\def \_underbracket #1{\_mathop {\_Umathaccent bottom 7 1 "023B5{#1}}\_limits}
\public \overbrace \underbrace \overparen \underparen \overbracket \underbracket ;
Aliases are declared here. They are names not mentioned in the \unimath-table.opm file but commonly used in TeX.
The \not macro is redefined here. If the \not\langle char \rangle is defined (by \negationof) then this macro is used. Else centered / is printed over the \langle char \rangle.

\protected\def\not#1{\protect\ifnum#1=0\protect\mathrel{\varTheta}%
\else\protect\mathrel{\mathstyles{\textstyle\setbox0=\hbox{$\currstyle\math$}{#1}}%\hbox to\wd0\hss\hss\kern-\wd0\box0}\fi}}
\def\negationof#1#2{\ea\let\csname\not!\csstring#1\endcsname=#2}

\negationof=\neq\negationof<\nless\negationof>\ngtr\negationof\gets\nleftarrow\negationof\simeq\nsime\negationof\equal\ne\negationof\le\nleq\negationof\ge\ngeq\negationof\greater\ngtr\negationof\forksnot\forks\negationof\in\notin\negationof\mid\nmid\negationof\parallel\nparallel\negationof\leftarrow\nleftarrow\negationof\leftrightarrow\nleftrightarrow\negationof\Rightarrow\nRightarrow\negationof\exists\nexists\negationof\ni\nni\negationof\approx\napprox\negationof\equiv\nequiv
507 \negationof \asymp \nasymp
508 \negationof \lessim \nlessim
509 \negationof \ngtrsim \ngtrsim
510 \negationof \lessgtr \nlessgtr
511 \negationof \gtless \ngtless
512 \negationof \prec \nprec
513 \negationof \succ \nsucc
514 \negationof \subset \nsubset
515 \negationof \supset \nsupset
516 \negationof \subseteq \nsubseteq
517 \negationof \supseteq \nsupseteq
518 \negationof \vdash \nvdash
519 \negationof \vDash \nvDash
520 \negationof \Vdash \nVdash
521 \negationof \VDash \nVDash
522 \negationof \preccurlyeq \npreccurlyeq
523 \negationof \succcurlyeq \nsucccurlyeq
524 \negationof \sqsubseteq \nsqsubseteq
525 \negationof \supseteq \nsupseteq
526 \negationof \vartriangleleft \nvartriangleleft
527 \negationof \vartriangleright \nvartriangleright
528 \negationof \trianglelefteq \ntrianglelefteq
529 \negationof \trianglerighteq \ntrianglerighteq
530 \negationof \vinfty \nvinfty
531 \public \not ;

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.

540 \_private
541 \ldotp \cdotp \bullet \triangleleft \triangleright \mapstochar \rightarrow
542 \prime \lhook \rightarrow \leftarrow \rhook \triangleright \triangleleft
543 \rbrace \lbrace \Relbar \Rightarrow \relbar \rightarrow \Leftarrow \mapstochar
544 \longrightarrow \Longleftrightarrow \unicodevdots \unicodedots ;

\_public \not ;

\_negationof \not ;

2.16.3 More Unicode-math examples

Example of using additional math font is in section 5.3 in the optex-math.pdf documentation More examples are in the OpTeX tricks and in the math.opm package.


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 T\v{E}X 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.

\_codeend \_undefined {Printing Unicode-math table \string<2020-06-08>}
5 \\_ifs\_ncharrmA \_undefined \_opwarning{No Unicode math font loaded, printing ignored}
6 \_endinput \_fi
7 \begin{group}
9 \_def\UnicodeMathSymbol#1#2#3#4{\
10 \_ifnum#1>"10000 \_endinput \_else \_printmathsymbol{#1}{#2}{#3}{#4}\_fi
11 }
12 \_def\UnicodeMathSymbolA#1#2#3#4{\
13 \_ifnum#1>"10000 \_printmathsymbol{#1}{#2}{#3}{#4}\_fi
14 }
15 \_def\printmathsymbol\#1\#2\#3\#4{\%\n16 \_bbox{\_bbox toem{\$#2\$\_hbox{\_bbox toem{\{small\_printop#3\_hss}{\tt\string#2\_trycs\{eq:\_string#2\}(}}}}}}
17 \_def\eq\#1\#2\_sdef\{eq:\_string#2\}
18 \_eq \diamond\smwhtdiamond \_eq \bullet\smblkcircle \_eq \circ\vysmwhtcircle
19 \_eq \diamond\smwhtdiamond \_eq \bullet\smblkcircle \_eq \circ\vysmwhtcircle

109
2.17 Scaling fonts in document (high-level macros)

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

\texttt{\textbackslash typosize \{[font-size]/\textbackslash baselineskip\}\} sets given parameters. It sets text size by the \texttt{\setfontsize} macro and math font sizes by setting internal macros \texttt{\sizemtext, \sizemscript} and \texttt{\sizemsscript}. It uses common concept font sizes: 100\%, 70\% and 50\%. The \texttt{\setmainvalues} sets the parameters as main values when the \texttt{\typosize} is called first.

\texttt{\textbackslash typoscale \{[font-factor]/\textbackslash baseline-factor\}\} scales font size and baselineskip by given factors in respect to current values. It calculates the \texttt{\typosize} parameters and runs the \texttt{\typosize}.
\_setbaselineskip \{baselineskip\} sets new \baselineskip and more values of registers which are dependent on the \baselineskip including the \strutbox.

\_setmainvalues sets the current font size and \baselineskip values to the \mainfontsize and \mainbaselineskip registers and loads fonts at given sizes. It redefines itself as \_setmainvaluesL to set the main values only first. The \_setmainvaluesL does only fonts loading. \ SCALEMAIN returns to these values if they were set. Else they are set to 10/12pt.

\_mfontsrule gives the rule how math fonts are loaded when \_typosize or \_typoscale are used. The value of \_mfontsrule can be:

- 0: no math fonts are loaded. User must use \_normalmath or \_boldmath explicitly.
- 1: \_normalmath is run if \_typosize/\_typoscale are used first or they are run at outer group level. No \_everymath/\_everydisplay are set in this case. If \_typosize/\_typoscale are run repeatedly in a group then \_normalmath is run only when math formula occurs. This is done using \_everymath/\_everydisplay and \_setmathfonts. \_mfontsrule=1 is default.
- 2: \_normalmath is run whenever \_typosize/\_typoscale are used. \_everymath/\_everydisplay registers are untouched.

Suppose following example: \{\_typosize[13/15] Let $M$ be a subset of $R$ and $x \in M$...\} 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.

\everymath \normalmath \everydisplay

\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.

\thefontsize \[size\] \thefontscale \[factor\]

\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.

\em \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.

\boldify

\rmfixed is used in default \footline

\rmfixed \[factor\] used in default \footline

\global\let\rmfixed=\tenrm next use will be font switch only

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 \typosize command and redefines itself be only the font switch for the next pages.)
2.18 Output routine

The output routine \_optexoutput is similar as in plain \TeX. It does:

- \_begoutput which does:
  - increments \_gpageno,
  - prints \_Xpage{\_gpageno} to the .ref file (if \openref is active),
  - calculates \hoffset,
  - sets local meaning of macros used in headlines/footlines (see \regmacro).
- \_optexshipout \_completepage, which is \vbox of –
  - background box, if \pgbackground is non-empty,
  - headline box by \_makeheadline, if the \headline is nonempty,
  - \vbox to\vsiz of \_pagecontents which consists of –
    - \_pagedest, the page destination pg;{\_gpageno} 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 0pt).
  - footline box by \_makefootline, if the \footline is nonempty
- \_endoutput which does:
  - increments \_pageno using \advancepageno
  - runs output routine repeatedly if \dosupereject is activated.

\_optexoutput is the default output routine. You can create another
\_begoutput={...} or \addto\_begoutput{...}
\_endoutput={...} or \addto\_endoutput{...}

The settings here are local in the \output group.

The \_prepoffsets can set \hoffset differently for the left or right page. It is re-defined by the \margins macro.

The \_regmark tokens list includes accumulated #2 from the \regmacro. Logos and other macros are re-defined here (locally) for their usage in headlines or footlines.

The \_optexshipout does similar work like the \_shipout primitive. The color literals are added to the \box0 using the \_preshipout(destination box number)\{box specification\} pseudo-primitive. It is defined using lua code, see section 2.39. Finally the \_shipout primitive is used.

The \_hsizvalue can be changed at various places in the document but we need to have a constant value \_xsiz in the output routine (for headlines and footlines, for instance). This value is set from the current value of \_hsiz when \_setxsiz 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 \_hsiz value is eventually changed by the user in this environment.
The \_completepage is similar to what plain \TeX does in its output routine. New is only \_backgroundbox. It is a vbox with zero height with its contents (from \_pgbackground) extended down. It is shifted directly to the left-upper corner of the paper.

The \_resetattrs used here means that all newly created texts in output routine (texts used in headline, footline) have default color and no transparency.

\_makeheadline creates a vbox to0pt with its contents (the \_headline) shifted by \_headlinedist up.

\_makefootline appends the \_footline to the page-body box.

The \_pagecontents is similar as in plain \TeX. The only difference is that the \_pagedest is inserted at the top of \_pagecontents.

The \_footnoterule is defined here.
Macros for footnotes are the same as in plain \TeX. There is only one difference: \texttt{\vfootnote} is implemented as \texttt{\_opfootnote} with empty parameter \texttt{#1}. This parameter should do local settings inside the \texttt{\footins} group and it does it when \texttt{\fnote} macro is used.

The \texttt{\_opfootnote} nor \texttt{\vfootnote} don't take the footnote text as a parameter. This is due to a user can do code settings (like inline verbatim) in the footnote text. This idea is adapted from plain \TeX.

The \texttt{\footnote} and \texttt{\footstrut} is defined as in plain \TeX.

\begin{verbatim}
\_newinsert\_footins
\_def \_footnote #1\{\_let\_osf=\_empty % parameter \texttt{#2} (the text) is read later
\_ifhmode \_edef\_osf{\_spacefactor\_the\_spacefactor}{\_fi
\_osf\_vfootnote{#1}
\_def \_vfootnote{\_opfootnote{}}
\_def \_opfootnote #1\{\_insert\_footins\_bgroup
\_interlinepenalty=\_interfootnotelinepenalty
\_leftskip=\_zo \_rightskip=\_zo \_spaceskip=\_zo \_xspaceskip=\_zo \_relax
\_resetattrs
\_fnote\{#1\relax % local settings used by \texttt{\fnote} macro
\_splittopskip=\_ht\_strutbox % top baseline for broken footnotes
\_splitmaxdepth=\_dp\_strutbox \_floatingpenalty=20000
\_textindent{#2}\_footstrut
\_isnextchar \_bgroup
{\_bgroup \_aftergroup\_vfootA \_afterassignment\_ignorespaces \_let\_next=}{\_vfootB}%
\_def \_vfootA{\_unskip\_strut\_egroup}
\_def \_vfootB #1{#1\_unskip\_strut\_egroup}
\_def \_footstrut {\_vbox to\_splittopskip{}}
\_skip\_footins=\_bigskipamount % space added when footnote is present
\_count\_footins=1000 % footnote magnification factor (1 to 1)
\_dimen\_footins=8in % maximum footnotes per page
\_public
\_footins \_vfootnote \_footstrut ;
\end{verbatim}

The \texttt{\topins} macros \texttt{\topinsert}, \texttt{\midinsert}, \texttt{\pageinsert}, \texttt{\endinsert} are the same as in plain \TeX.

\begin{verbatim}
\_newinsert\_topins
\_newifi\_ifupage \_newifi\_ifumid
\_def \_topinsert {\_umidfalse \_upagefalse \_oins}
\_def \_midinsert {\_umidtrue \_oins}
\_def \_pageinsert {\_umidfalse \_upagetrue \_oins}
\_skip\_topins=\_zoskip % no space added when a topinsert is present
\_count\_topins=1000 % no limit per page
\_dimen\_topins=\_maxdimen % no limit per page
\_def \_oins {\_par \_begingroup \_setbox0=\_vbox \_resetattrs}
\_def \_endinsert {\_par \_egroup}
\_ifumid \_dimen0=\_ht0 \_advance\_dimen0 by\_dp0 \_advance\_dimen0 by\_baselineskip
\_ifupage \_dimen0=\_dp0
\_vbox to\_vsize {\_unvbox0 \_kern-\_dimen0}% depth is zero
\_else \_box0 \_nobreak \_bigskip \_fi \_fi
\_ifumid \_bigskip \_box0 \_bigbreak
\_else \_insert \_topins {\_penalty100 % floating insertion
\_splittopskip=\_opt
\_splitmaxdepth=\_maxdimen \_floatingpenalty=0
\_ifupage \_dimen0=\_dp0
\_vbox to\_vaize {\_unvbox0 \_kern-\_dimen0}\_depth is zero
\_else \_box0 \_nobreak \_bigskip \_fi \_fi\_endgroup}
\_ifdef\_draft\{\_draftbox\}
\_ifndef\_draft\{\_draftfont\}
\_global\_let\_draftfont=\_draftfont
\}
\_def \_draftbox #1\{\_boxed{\_setgreycolor{.8}#1}\}
\_ifdef\_draft\{\_draftfont\}
\_ifndef\_draft\{\_setfontsize{at10pt}\_bf\}
\_global\_let\_draftfont=\_draftfont
\}
\end{verbatim}

The \texttt{\draft} macro is an example of usage \texttt{\_pgbackground} to create watercolor marks.

\begin{verbatim}
\_def \_pgbackground=\_draftbox\{\_draftfont DRAFT\}
\_fontdef\_draftfont{\_setfontsize{at10pt}\_bf}
\_global\_let\_draftfont=\_draftfont
\}
\_def \_draftbox #1\{\_boxed{\_setgreycolor{.8}#1}\}
\_ifdef \_kern\.5\_vaize \_kern\_offset \_kern4.5\_wd0
\_bbox to\_opt{\_kern\_5\_xsize \_kern\_hoffset \_kern-2\_wd0
\_pdfsave \_pdfrotate(55) \_pdfscale{10}{10} \_pdfrestore
\_bbox to\_opt{\_box0\_has}%
\end{verbatim}

\section*{115}
2.19 Margins

The \margins macro is documented in the section 1.2.1.

\margins\{<pag>\{fnt\}\{left\},\{right\}\{top\},\{bot\}\{unit\}\" takes its parameters, does calculation and sets \hoffset, \voffset, \hsize and \vsize registers. Note that OpTEX sets the page origin at the top left corner of the paper, no at the obscure position 1in, 1in. It is much more comfortable for macro writers.

The common page dimensions are defined here.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>a3</td>
<td>(297, 420) mm</td>
</tr>
<tr>
<td>a4</td>
<td>(210, 297) mm</td>
</tr>
<tr>
<td>a5</td>
<td>(148, 210) mm</td>
</tr>
<tr>
<td>a3l</td>
<td>(420, 297) mm</td>
</tr>
<tr>
<td>a4l</td>
<td>(297, 210) mm</td>
</tr>
<tr>
<td>a5l</td>
<td>(210, 148) mm</td>
</tr>
<tr>
<td>b5</td>
<td>(176, 250) mm</td>
</tr>
<tr>
<td>letter</td>
<td>(8.5, 11) in</td>
</tr>
</tbody>
</table>

\magscale \{<factor>\} does \magscale \{<factor>\} and recalculates page dimensions to their true values.
When left-to-right direction of typesetting is selected (default) then “main vertical line” of the page has \hoffset distance from the left paper border and all lines at the page start here and run to the right side (exceptions can be done by \moveleft or \overright, of course). When we have set right-to-left direction (using \textdir TRT, for example), then the “main vertical line” cannot be at the same position because lines run to the left, i.e. they would be off paper. This is reason why the setting \pagedir TRT shifts the “main vertical line” to an alternative position: it has \pagerightoffset+1in distance from the right paper border and thus right-to-left lines are visible on the paper. We have to set \pagerightoffset properly for such cases. This is done in the macro \_setpagerightoffset. It must be called whenever \hoffset is changed.

\_setpagerightoffset\% setting default value from default values

Page numbers and numbers of (sub)sections have to be printed in left-to-right mode even though the document mode is right-to-left. We print these numbers via \_numprint{⟨number⟩} in OpTEX macros. The \_numprint is \_useit by default (i.e. do nothing special) because we have left-to-right mode as default. But a user can define

\_def\_numprint#1{{\textdir TLT #1}}

if the document is set to right-to-left mode.

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 whatsis (either using \special for DVI or using \pdfliteral/\pdfcolorstack 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 \pdfcolorstack to Lua\TeX 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 \inserts, etc. Moreover, once a material is created (using \setbox for example) then it has its fonts and its colors frozen and you can rely on it when you are using e.g. \unhbox. There are no internal whatsis for colors which can interfere with other typesetting material. In the end something like setting text to red (\textcolor{Red}{text}) should have the same nice behavior like setting text to bold (\textbf{text}).

Lua\TeX attributes can be set like count register – one attribute holds one number at a 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 \_colorattr for storing the currently selected color (in number form).

Macros \setcmykcolor{⟨C⟩ ⟨M⟩ ⟨Y⟩ ⟨K⟩} or \setrgbcolor{⟨R⟩ ⟨G⟩ ⟨B⟩} or \setgreycolor{⟨Grey⟩} are used in color selectors. These macros expand to internal \_setcolor macro which sets the \_colorattr attribute to an integer value and prepares mapping between this value and

117
the real color data. This mapping is used just before each \shipout in output routine. The \_preshipout pseudo-primitive is used here, it converts attribute values to internal PDF commands for selecting colors.

The concept with color attributes has one limitation: the colors cannot be changed inside a ligature unless the ligature is broken manually. It means that \texttt{\{\Red f\}i} doesn’t lead to the expected result but \texttt{\{\Red f\ null\}i} does.

### 2.20.2 Color mixing

The color mixing processed by the \colordef 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 toners and the result is more true. This should be done by \useK command at the end of a linear combination used in \colordef. For example

\colordef \myColor {.3\Green + .4\Blue \useK}

The \useK command exactly does:

\[
k' = \min(C, M, Y),
C = (C - k')/(1 - k'),
M = (M - k')/(1 - k'),
Y = (Y - k')/(1 - k'),
K = \min(1, K + k').
\]

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 \{-\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 preceded 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):

#### CMYK to RGB:

\[
\]

#### RGB to CMYK:

\[
K' = \max(R, G, B), \quad C = (K' - R)/K', \quad M = (K' - G)/K', \quad Y = (K' - B)/K', \quad K = 1 - K'.
\]

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 \_codedecl \colordef \{Colors <2022-03-07>} % preloaded in format

The basic colors in CMYK \Blue \Red \Brown \Green \Yellow \Cyan \Magenta \Grey \LightGrey \White and \Black are declared here.

118
By default, the \setcmykcolor, \setrgbcolor, and \setgreycolor macros with \{⟨components⟩\} parameter expand to \_setcolor{⟨color-data⟩}{⟨fill-op⟩}{⟨stroke-op⟩} where \{color-data\} is \{R\} \{G\} \{B\} or \{C\} \{M\} \{Y\} \{K\} or \{ ⟨Y⟩ \{K⟩ or \{⟨G⟩ and \{fill-op⟩ is color operator for filling, \{⟨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.}

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⟩

% Black is the default color.
\sdef\_color::0 g{0}
\sdef\_color:0 {0 g}
\sdef\_color-s:0 {0 G}

119
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.

The attribute \_transpattr is allocated and set by the \transparency⟨number⟩ macro. If such level of the transparency was never used in the document then \addextgstate{tr⟨number⟩} is applied (where X is (255−⟨number⟩)/255). This information is used when shipout is processed (similarly as colors). It means /tr⟨number⟩gs is inserted when the attribute is changed.

\_resetattrs resets the \_colorattr and \_transpattr to their initial value −"7FFFFFFF. We use Lua codes for RGB to CMYK or CMYK to RGB conversions and for addition color components in the \colordef macro. The \_rgbtocmyk ⟨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 \rgbcmykmap) 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 0.400 0.560 ; expands to .3 .4 .56.

\begin{verbatim}
\def\_stripzeros\#1\#2{\if\ifx^\#2^\_stripzeroC\#1:\_else \_stripzeroB\#1 0 :\fi}
\def\_stripzeroA#10 #2:#{\_ifx^#2^\_stripzeroC#1:\_else \_stripzeroB#1 0 :\fi}
\def\_stripzeros #1.#2 #3\fi{\_ifx0#1\_else#1.\_stripzeroA #2 0 :%}
\end{verbatim}

\texttt{\_rgbcolordef} and \texttt{\_cmykcolordef} use common macro \texttt{\_commoncolordef} with different first four parameters. The \texttt{\_commoncolordef} \{\texttt{selector}\}\{\texttt{K}\}\{\texttt{R}\}\{\texttt{G}\}\{\texttt{what-define}\}\{\texttt{data}\} does the real work. It initializes the Lua variables for summation. It expands \{\texttt{data}\} in the group where color selectors have special meaning, then it adjusts the resulting string by \texttt{\_ replstring} and runs it. Example shows how the \{\texttt{data}\} are processed:

\begin{verbatim}
input \{data\}: \".3\Blue \+.6\KhakiC \useK \-\Black\"
expanded to: \".3 !=K 1 1 0 0 \+.6\!=-R .804 .776 .45 \_useK \!=-G 0\"
adjusted to: \"\_addcolor \-.3!=K 1 1 0 0 \_addcolor \+.6!=R .804 .776 .45 \_useK \_addcolor \-1!=G 0\"
and this is processed.
\end{verbatim}

\texttt{\_addcolor} \{\texttt{coef}\}!\{\texttt{mod}\}!\{\texttt{type}\} expands to \texttt{\_addcolor}:\{\texttt{mod}\}:\{\texttt{type}\} \{\texttt{coef}\} for example it expands to \texttt{\_addcolor}:K \{\texttt{coef}\} followed by one or three or four numbers (depending on \{\texttt{type}\}). \{\texttt{mod}\} is = \{use as is\} or \(\neq\) \{use complementory color\}. \{\texttt{type}\} is \texttt{K} for CMYK, \texttt{R} for RGB and \texttt{G} for GREY color space. Uppercase \{\texttt{type}\} informs that \texttt{\_cmykcolordef} is processed and lowercase \{\texttt{type}\} informs that \texttt{\_rgbcolordef} is processed. All variants of commands \texttt{\_addcolor}:\{\texttt{mod}\}:\{\texttt{type}\} are defined. All of them expand to \texttt{\_addcolor\#1} \{\texttt{ef}\} \{\texttt{ef}\} \{\texttt{ef}\} \{\texttt{ef}\} which adds the values of Lua variables. The \texttt{\_rgbcolordef} uses \texttt{\_addcolor\#1} \{\texttt{R}\} \{\texttt{G}\} \{\texttt{B}\} 0 and \texttt{\_cmykcolordef} uses \texttt{\_addcolor\#1} \{\texttt{C}\} \{\texttt{M}\} \{\texttt{Y}\} \{\texttt{K}\}. So the Lua variable names are a little confusing when \texttt{\_rgbcolordef} is processed.

Next, \texttt{\_commoncolordef} saves resulting values from Lua to \texttt{\_tmpb} using \texttt{\_colordefFin}. If \texttt{\_rgbcolordef} is processed, then we must to remove the last \{\texttt{K}\} component which is in the format .0 in such case. The \texttt{\_stripK} macro does it. Finally, the \{\texttt{what-define}\} is defined as \{\texttt{selector}\}\{\texttt{expanded \_tmpb}\}, for example \texttt{\_setcmykcolor\{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:

\begin{verbatim}
def\vr{\vrule height10pt depth2pt width20pt}
def\showcolor{\hbox{\tt\_bslash\_tmpb: \csname\_tmpb\endcsname \vr}\space\space}
begmulti 4 \typsize[10/14]
\morecolors
\endmulti
\end{verbatim}
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{⟨gpageno⟩}{⟨pageno⟩}
\_Xtoc{⟨level⟩}{⟨type⟩}{⟨text⟩}{⟨title⟩}
\_Xlabel{⟨label⟩}{⟨text⟩}
\_Xpage{⟨gpageno⟩}{⟨pageno⟩}
\_Xlabel{⟨label⟩}{⟨text⟩}
...

- \_Xpage corresponds to the beginning of a page. ⟨gpageno⟩ 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).

The \_inputref macro is executed in \everyjob. It reads the \jobname.ref file, if it exists. After the file is read then it is removed and opened for writing.

\_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 \optex op-demo twice to see the effect.

If the .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 \jobname.ref file is created (it would be unusable). The \_wref macro is a dummy in that case.
If a macro needs to create and use the .ref file, then such macro must first use `\openref`. It creates the file and redefines \_wref \langle macro{\{data\}}\rangle so that it saves the line \langle macro{\{data\}}\rangle to the .ref file using the asynchronous `\write` primitive. Finally, \_openref destroys itself, because we don’t need to open the file again.

\_wref\langle csname{\{params\}}\rangle in fact does `\write\_reffile{\string\langle csname{\{params\}}\rangle}` and similarly \_ewref\langle csname{\{params\}}\rangle does `\write\_reffile{\string\langle\{expanded-params\}\rangle}`.

We are using the convention that the macros used in .ref file are named \_X\langle foo\rangle. We don’t want to read .ref files from old, incompatible versions of OpTEX (and OPmac). This is ensured by using a version number and the `\Xrefversion` macro at the beginning of the .ref file:

`\Xrefversion\{\langle version\rangle\}`

The macro checks the version compatibility. Because OPmac does not understand `\Xrefversion` we use `\Xrefversion` (with a different number of \langle version\rangle than OPmac) here. The result: OPmac skips .ref files produced by OpTEX and vice versa.

You cannot define your own .ref macros before .ref file is read because it is read in `\everyjob`. But you can define such macros by using `\refdecl{\langle definitions of your ref macros\rangle}`. This command writes \langle definitions of your ref macros\rangle to the .ref file. Then the next lines written to the .ref file can include your macros. An example from CTUstyle2:

\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}}}}

We must read \langle definitions of your ref macros\rangle while \# has the catcode 12, because we don’t want to duplicate each \# in the .ref file. \refdecl appends its data to the \_refdecldata macro. It is pushed to the .ref file immediately only if the file is opened already. Otherwise we are waiting to `\openref` because we don’t want to open the .ref file if it is unnecessary.

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.
Counter for the number of unresolved references \_unresolvedrefs. It is set but unused in OpTeX versions 1.04+. You can add the report, for example:

\_addto\_byehook{\_ifnum\_unresolvedrefs>0 \_opwarning {There are \_the\_unresolvedrefs\_space unresolved references}\_fi}

\_newcount\_unresolvedrefs
\_unresolvedrefs=0

\_label{⟨label⟩}{⟨text⟩} saves the ⟨text⟩ to \_lab:⟨label⟩ and saves ⟨⟨gpageno⟩⟨pageno⟩⟩ to \_pgref:⟨label⟩.

\_def\_label[#1]{\_isempty{#1}\_iftrue \_global\_let \_lastlabel=\_undefined \_else \_isdefined{l0:#1}% \_iftrue \_slideshook\_opwarning{Duplicated label \[#1\], ignored}\_else \_xdef\_lastlabel{#1}\_fi \_fi}

\_let \_slideshook=\_relax % redefined if \slides + \slideshow.
\_def\_wlabel#1{\_ifx\_lastlabel\_undefined \_else\_dest[ref:\_lastlabel]{\_printlabel\_lastlabel}\_ewref \_Xlabel {{\_lastlabel}{#1}}\_sxdef{l0:\_lastlabel}{}\_global\_let\_lastlabel=\_undefined\_fi\_fi}

\_public \label \wlabel ;
\_ref[⟨label⟩]{⟨given-text⟩} prints (linked) ⟨given-text⟩. The missing optional ⟨⟨given-text⟩⟩ is replaced by \@. The \@ is replaced by ⟨⟨implicit-text⟩⟩ from saved \lab:⟨label⟩ using \_reftext macro. If the reference is backward then we know \lab:⟨label⟩ without any need to read REF file. On the other hand, if the reference is forwarded, then we doesn’t know \lab:⟨label⟩ in the first run of \TeX and we print a warning and do \_openref.
\_pgref[⟨label⟩]{⟨given-text⟩} prints ⟨given-text⟩ where \@ is replaced by ⟨⟨pageno⟩⟩. Data in the format ⟨⟨gpageno⟩⟨pageno⟩⟩ are read from \_pgref:⟨label⟩ by \_pgrefB{⟨⟨pagn}o⟩}{⟨⟨pagn}o⟩}{⟨given-text⟩}. \_lastreflabel keeps the value of the last label read by \ref or \pgref. You can use it for example by defining a macro \_pgr by \def\pgr{\pgref[\_lastreflabel]} and then you need not repeat the same label in typical situations and you can write for instance: see section \ref[lab] at page \pgr.
The \ref{cmd} \verb!\ref! expands to the \verb!\ref! but the optional \verb!\texttt{} in the \verb!\ref! is replaced by the \verb!\texttt{} first.

\verbatiminput{references.oml}

Default \verb!\printlabel! is empty macro (labels are not printed). The \verb!\showlabels! redefines it as box with zero dimensions and with left lapped \verb!\label{}! in blue 10pt \verb!\tt! font shifted up by 1.7ex. The color of labels is set by \verb!\labelcolor! (default is RGB blue).

\verbatiminput{references.oml}

\subsection{Hyperlinks}

There are six types of internal links and one type of external link used in OpTEX. They are used in the format \verb!\type{spec}!.

- \verb!\ref{label}! – the destination is created when \verb!\label{label}! is used, see also the section 2.22.
- \verb!\toc{tocrefnum}! – the destination is created at chap/sec/sec titles, see also the section 2.24.
- \verb!\pg{gpage}! – the destination is created at beginning of each page, see also the section 2.18.
- \verb!\cite{bibpart}/\bibnum{bibnum}! – the destination is created in bibliography reference, see section 2.32.1.
- \verb!\nt{gnfnotenum}! – link form text to footnote, see also section 2.34.
- \verb!\ntf{gntfnotenum}! – link from footnote to text, see also section 2.34.
- \verb!\url{url}! – used by \verb!\url! or \verb!\ulink!, see also the end of this section.

The \verb!\toc{tocfnum}!, \verb!\gpage{gpage}!, \verb!\bibnum{bibnum}!, and \verb!\gnfnotenum{}! are numbers starting from one and globally incremented by one in the whole document. The registers \verb!\tocfnum{}, \verb!\gpage{}, \verb!\bibnum{}, and \verb!\gnfnotenum{}! are used for these numbers.

When a chap/sec/sec title is prefixed by \verb!\label{label}!, then both types of internal links are created at the same destination place: \verb!\toc{tocfnum}! and \verb!\ref{label}!.

The color for active links can be declared by \verb!\def\_(\type{type})linkcolor{}! macro and borders are invisible. For example \verb!\def\_toclinkcolor! \verb!\Red{}! means that links from table of contents are in red. Another example \verb!\def\_tocborder! \verb!(1 0 0)! causes red frames in TOC (not printed, only visible in PDF viewers).

\verbatiminput{hyperlinks.oml}

\verbatiminput{hyperlinks.oml}

Each hyperlink is created internally by \verb!\xlink{\type{type}}{\spec{spec}}{\color{color}}{\text{text}}{}! This macro expands to \verb!\xlink{}{}{}! by default, i.e. no active hyperlink is created, only \verb!\text{text}! 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 \text{⟨text⟩} has given \text{⟨color⟩} only when hyperlink is created. If \text{⟨⟨type⟩⟩linkcolor} is defined, it has precedence over \text{⟨color⟩}.

The \linkdimensions macro declares the dimensions of link area.

A specific action can be defined for each link \text{⟨type⟩} by the macro \text{⟨⟨type⟩⟩action}({\spec}). Op\LaTeX\ defines only \urlaction{⟨url⟩}. The default link action (when \text{⟨⟨type⟩⟩action} is not defined) is \text{goto name}({\text{⟨⟨type⟩⟩}: {⟨spec⟩}}) (an internal link). It is declared in the \text{⟨⟨type⟩⟩linkactions}({⟨spec⟩}) macro. The \pdfstartlink primitive uses \text{attrib}{}\text{⟨pdfborder{⟨type⟩}⟩}. The \text{pdfborder⟨⟨type⟩⟩} macro expands to \text{⟨/C[? ? ?]} \text{⟨Border[0 0 .6]} if the \text{⟨⟨type⟩⟩border} macro (i.e. \text{⟨refborder, _citeborder, _tocborder, _pgborder, _urlborder, _fntborder or _fnfborder⟩}) is defined.

\text{\lbrack⟨⟨type⟩⟩: {⟨spec⟩}}\{⟨color⟩}\{⟨text⟩\} creates a link. It is kept here for backward compatibility and it is equivalent to \text{\lbrack⟨⟨type⟩⟩: {⟨spec⟩}}\{⟨color⟩\}\{⟨text⟩\}. If \text{⟨⟨type⟩⟩action} is not defined then \url action creates internal link do the \text{\dest{⟨⟨type⟩⟩: {⟨spec⟩}}}]. You can have more links with the same \text{⟨⟨type⟩⟩: {⟨spec⟩}} but only one \text{\dest} in the document.

\text{\lbrack⟨⟨type⟩⟩: {⟨spec⟩}}\{⟨text⟩\} creates external link. The \text{⟨url⟩} is detokenized with \text{escapechar=-1} before it is used, so \text{\#, \#} etc. can be used in the \text{⟨url⟩}.

\hyperlinks{⟨ilink color⟩}{⟨ulink color⟩} activates \text{\dest, \xlink}, so that they create links. Not setting colors \text{⟨hyperlinks⟩} is also supported.

\text{⟨url⟩} does approximately the same as \text{⟨ulink⟩}{⟨url⟩}, but more work is done before the \url action is processed. The link-version of \text{⟨url⟩} is saved to \text{\tmpa} and the printed version in \text{\tmpb}. The printed version is processed in four steps: 1. the \text{\1} are replaced by \text{[11]} (we suppose that such string does not exist in any URL). 2. it is detokenized with \text{escapechar=-1}. 3. multi-strings and spaces are replaced by strings in braces { ...}. 4. internal penalties and skips are put between characters using \text{\urlA, \urlB and \urlC}. The step 4 do following: The \text{\urllskip} is inserted between each pair of “normal characters”, i.e. characters not declared by \text{\def{ur:(character)}}. The special characters declared by \text{\def{ur:(character)}} are replaced by the body of their corresponding macro. The \text{\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

\begin{verbatim}
\protected\def\xlinkactive#1#2#3#4(\quitzvmode
\pdfstartlink\linkdimensions attr{\pdfborder{#1}}\linkactions{#1}{#2}\relax
\localcolor\trycs{\#1linkcolor}{#3}#4}
\protected\def\xlinkcolor#1#2{\let\xlinkcolor=#1\empty}
\let\destactive\xlinkcolor
\let\linkcolor\xlinkcolor
\def\pdfborder#1(\ifsname \action \endsname \lastnamedcs{#2}\else goto name{#1:#2}\fi)
\def\linkactions#1#2{\ifcsname _#1action\endcsname}
\def\hyperlinks#1#2#3#4{\noexpand\xlink{url}{\detokenize{#1}}\elinkcolor{#2}}
\def\url{⟨url⟩}{⟨url⟩} does approximately the same as \text{\lbrack⟨⟨type⟩⟩: {⟨spec⟩}}\{⟨color⟩\}\{⟨text⟩\}, but more work is done before the \url action is processed. The link-version of \text{⟨url⟩} is saved to \text{\tmpa} and the printed version in \text{\tmpb}. The printed version is processed in four steps: 1. the \text{\1} are replaced by \text{[11]} (we suppose that such string does not exist in any URL). 2. it is detokenized with \text{escapechar=-1}. 3. multi-strings and spaces are replaced by strings in braces { ...}. 4. internal penalties and skips are put between characters using \text{\urlA, \urlB and \urlC}. The step 4 do following: The \text{\urllskip} is inserted between each pair of “normal characters”, i.e. characters not declared by \text{\def{ur:(character)}}. The special characters declared by \text{\def{ur:(character)}} are replaced by the body of their corresponding macro. The \text{\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

\end{verbatim}
normal characters, penalty 100 is inserted after special characters, 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_urlxskip=\nobreak. The text version of the ⟨url⟩ is printed in \_urlfont.

\begin{verbatim}
\def\_urlfont{\tt} % url font
\def\_urlxskip{\penalty9990\hskip0pt plus0.03em\relax} % skip between normal characters
\def\_urlskip{\null\nobreak\hskip0pt plus0.1em\relax} % skip before :// / . ? = - &
\def\_urlbskip{\penalty100\hskip0pt plus0.1em\relax} % skip after :// / . ? = - &
\def\_urlgskip{\penalty-500\relax} % "goodbreak" penalty generated by \|

\public\url; \end{verbatim}

2.24 Making table of contents

\begin{verbatim}
\def\_toclist{}\newifi\ifischap\ischapfalse
\newifi\ifnum\level=0\ischaptrue\fi
\newifi\ifnum\level=0\ischaptrue\fi
\newifi\ifnum\level=0\ischapfalse\fi
\def\_Xtoc\_level\_type\_number\_o-title\_title\_gpageno\_pageno{
\addtolist\_tocline\_level\_type\_number\_o-title\_title\_gpageno\_pageno}

\public \url; \end{verbatim}

The last two parameters are restored from previous \_Xpage\_pageno\_gpageno, data were saved in the \currpage macro.
We read the \title parameter by \scantoeol from .ref file because the \title can include something like `{`.\end{verbatim}

\begin{verbatim}
\def\_toclist{}\newifi\ifischap\ischapfalse
\newifi\ifnum\level=0\ischaptrue\fi
\newifi\ifnum\level=0\ischaptrue\fi
\newifi\ifnum\level=0\ischapfalse\fi
\def\_Xtoc\_level\_type\_number\_o-title\_title\_gpageno\_pageno{
\addtolist\_tocline\_level\_type\_number\_o-title\_title\_gpageno\_pageno}

\public \url; \end{verbatim}

128
\_tocline{⟨level⟩}{⟨type⟩}{⟨number⟩}{⟨o-title⟩}{⟨title⟩}{⟨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:

| ⟨number⟩ | text text text text text text text text ................................ | ⟨pageno⟩ |

Margins given by \_leftskip and \_rightskip are denoted by | in the example above. \tocrefnum is the global counter of all TOC records (used by hyperlinks).

You can re-define default macros for each level of tocline if you want. Parameters are {⟨number⟩}{⟨title⟩}{⟨pageno⟩}.

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 record with 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:

```latex
\def\addtotoc#1#2#3#4#5{%
  \incr\tocrefnum
  \dest[\_tocr:⟨title⟩]{\_xrefnumprint{#1}}\_kern.4em\_kern.1em%
  \_ewref\Xtoc{{#1}{#2}{#3}{#4}{#5}}%
}%
```

and you can declare special lines (or something else) as an unused level (10 in the following example):

\sdef{\_tocl:10}{#1#2#3\medskip\hbox{\Blue #2}\medskip}
Now, users can add a blue line into TOC by
\addtotoc{10}{blue-line}{relax}{⟨blue text to be added in the TOC⟩}
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.
Now, users can add a blue line into TOC by
\addtotoc{10}{blue-line}{relax}{⟨blue text to be added in the TOC⟩}
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.
\maketoc
\regmacro appends its parameters to \_regtoc, \_regmark and \_regoul. These token lists are used in \maketoc, \_begoutput and \pdfunidef.
\maketoc
\maketoc
\maketoc
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⟩:⟨num⟩ or 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
\_ol:\_count0:\_count1:\_count2 of this section
\advance \_ol:\_count0:\_count1 by 1
% 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.
\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

```
<FEFF 0043 0076 0069 010D 0065 006E 00ED 0020 006A 0065 0020 007A 00E1 0074 011B 017E 0020 0061 0020 0078 2208 D835DD44>
```

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
\_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 \_prepinverb{\_macro}{\_separator}{\_text} does \_macro as \_text converted to Big Endian UTF-16 and enclosed to <> . Example of usage: \_prepinverb\_infoauthor{Petr Olšák} \_pdfinfo{/Author \_infoauthor}.
It means that in-line verbatim are not argument of \texttt{scantextoken}. First \texttt{edef/\tmpb} tokenizes again the \texttt{text} but not the parts which were in the in-line verbatim.

The \texttt{regmacro} is used in order to set the values of macros \texttt{em}, \texttt{rm}, \texttt{bf}, \texttt{it}, \texttt{bi}, \texttt{tt}, \texttt{v} and - to values usable in PDF outlines.

You can re-define \texttt{\_printtit} macros if another design of section titles is needed. These macros get the \texttt{title} text in its parameter. The common recommendations for these macros are:

- Use \texttt{\_abovetitle}\{\texttt{\penalty}A\}\{\texttt{\skip}A\} and \texttt{\_belowtitle}\{\texttt{\skip}B\} for inserting vertical material above and below the section title. The arguments of these macros are normally used, i.e. \texttt{\_abovetitle} inserts \texttt{\penalty}A\{\texttt{\skip}A\} and \texttt{\_belowtitle} inserts \texttt{\skip}B. But there is an exception: if \texttt{\_belowtitle}\{\texttt{\skip}B\} is immediately followed by \texttt{\_abovetitle}\{\texttt{\penalty}A\}\{\texttt{\skip}A\} (for example section title is immediately followed by subsection title), then only \texttt{\skip}A is generated, i.e. \{\texttt{\skip}B\} is reduced only to \texttt{\skip}A. 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{\_abovetitle}\{\texttt{\penalty}A\}\{\texttt{\skip}A\} takes previous whatever vertical skip (other than from \texttt{\_belowtitle}) and generates only greater from this pair of skips. It means that \{\texttt{whatever-skip}\} is transformed to \texttt{\penalty}A\{\texttt{\max\{whatever-skip\}}\}\{\texttt{\skip}A\}. The reason for such behavior: we don’t want to duplicate vertical skips (from \texttt{\belowskip}, for example) above the title.
- Use \texttt{\_printrefnum}\{\texttt{\pre}@\texttt{\post}\} in horizontal mode. It prints \texttt{\pre}\{\texttt{\ref-num}\}\{\texttt{\post}\}. The \texttt{\ref-num} is \texttt{\thechapnum} or \texttt{\theseccnum} or \texttt{\thesectionnum} depending on what type o title is processed. If \texttt{\nonum} prefix is used then \texttt{\_printrefnum} prints nothing. The macro \texttt{\_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 \npar 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.

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 OpTEX trick 0033)

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 \fnotenumber are numbered in each chapter from one.
The \_notoc and \_nonum prefixes are implemented by internal \_ifnotoc and \_ifnonum. 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 \tite-text is \detokenize\d before \wref, so the problem of “fragile macros” from old \La\TeX\ 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.
The \_abovetitle{\{\penaltyA\}\{\penaltyB\}} and \_belowtitle{\{\penaltyB\}} 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 \{\penaltyA\} only if whatever previous skip is less or equal than \{\penaltyA\}. We must save \{whatever-skip\}, remove it, create \{\penaltyA\} (if \_belowtitle does not precede) and create \{whatever-skip\} or \{\penaltyA\} depending on what is greater. The amount of \{\penaltyA\} is measured using \setbox0=\vbox.

\nbpar sets \interlinepenalty value. \n1 is “new line” in the text (or titles), but space in toc or headlines or outlines.

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{\{\title text\}} saves \mark in the format \{\{\title-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.

OpTeX sets \headline={} by default, so no running headings are printed. You can activate the running headings by following code, for example. See also issue 100.
The `\usecl<number>{title-text}{eol}` should be used for various levels of sections (for example, when converting from Markdown to Opt(TEX)). `\secl` is `\chap`, `\secl2` is `\sec`, `\secl3` is `\secc` and all more levels (for `\langle 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.

```latex
\_def\_secl{\_afterassignment\_secla \_sectionlevel=}
\_def\_secla{\_ifcase\_sectionlevel
\_or\_ea\_chap\_or\_ea\_sec\_or\_ea\_secc\_else\_ea\_seclp\_fi}
\_edef\_seclp{\_par\_nobreak
\_vadjust{\_nobreak}\_nl\_ignorepars}
\_public \secl ;
```

The `\caption⟨letter⟩` increases `\langle letter⟩num` counter, edefines `\_thecapnum` as `\_the⟨letter⟩num` and defines `\_thecaptitle` as language-dependent word using `\_mtext`, declares default format by `\_captionformat{⟨letter⟩}` and runs the `\_everycaption{⟨letter⟩}` tokens register. The two groups opened by `\caption` are finalized by first `\_par` from an empty line or from `\vskip`, `\cskip` or from `\endinsert`. If an } occurs first then `\_par` from `\aftergroup` is processed. The `\_printcaption{⟨letter⟩}` is called, it starts with printing of the caption. The `\cskip` macro inserts nonbreakable vertical space between the caption and the object.

```latex
\_def\_caption/#1{\_def\_tmpa{#1}\_nospaceafter \_capA}
\_optdef\_capA[{}]{\_trylabel \_incaption}
\_def\_incaption{\_bgroup
\_ifcsname _\_tmpa num\_endcsname \_ea\_incr \_csname _\_tmpa num\_endcsname
\_else \_opwarning{Unknown caption /\_tmpa}\_fi}
\_edef\_thecapnum {\_csname _the\_tmpa num\_endcsname}\
\_edef\_thecaptitle{\_mtext{\_tmpa}}\
\_ea\_captionformat\_ea{\_tmpa}\
\_ea\_the \_csname _everycaption\_tmpa\_endcsname
\_ifx\par\_endgraf \_let\par=\_par \_fi\
\_bgroup \_aftergroup\_par
\_cs{\_printcaption\_tmpa}\
\_public \_caption \_cskip ;
```

The `\_printcaptiont` and `\_printcaptionf` macros start in vertical mode. They switch to horizontal mode and use `\_wlabel\_thecapnum` (in order to make reference and hyperlink destination). They can use:

- `\_thecaptitle` ... expands to the word Table or Figure (depending on the current language).
- `\_thecapnum` ... expands to `\_the⟨letter⟩num` (caption number).

The macro `\_printcaption{⟨letter⟩}` is processed inside group and the `\_par` can be run after this group. If you want to re-define formatting parameters for `\_par`, do this in the macro `\_captionformat`. 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.

```latex
\_def \_printcaption{\% \_noindent \_wlabel\_thecapnum \_bf\_thecaptitle\_thecapnum\%}
\_futurelet\_next\_captionsep
\_def\_captionsep{\_ifx\_next.\_ea\_bfnext \_else\_ifx\_next:\_ea\_ea\_ea\_bfnext
\_else \_enspace \_fi\_fi}
\_def\_bfnext#1{{\_bf#1}}
\_let \_printcaptionf = \_printcaption\% caption of figures = caption of tables
```

If you want to declare a new type of `\caption` with independent counter, you can use following lines, where `\caption/a` for Algorithms are declared:
The format of the \caption text is given by the \captionformat{⟨caption-letter⟩} macro. The default format for \texttt{t} and \texttt{f} is a paragraph in block narrower by \_iindent and with the last line centered. This setting is done by the \narrowlastlinecentered macro.

\eqmark is processed in display mode (we add \eqno primitive) or in internal mode when \equaligno is used (we don’t add \eqno).

The \numberedpar⟨letter⟩{⟨name⟩} is implemented here.

\begin{Verbatim}
\def\resetABCDE {\theA=0 \theB=0 \theC=0 \theD=0 \theE=0 }
\def\theAnum {\thechapnum.\thesection.\thenumber}
\def\theBnum {\thechapnum.\thesection.\thenumberB}
\def\theCnum {\thechapnum.\thesection.\thenumberC}
\def\theDnum {\thechapnum.\thesection.\thenumberD}
\def\theEnum {\thechapnum.\thesection.\thenumberE}
\def\numberedpar#1#2{\ea \ocr \theA \df \tmpa{#1}\df \tmpb{#2}\numberedparparam)
\def\numberedparparam[]\%
\def\ea \ocr \thenumber \thechapnum \thechapter \thenum \text{\numprint{\thenumber}}
\def\printnumberedpar \ocr \thenumber \text{\numprint{\thenumber}}
\def\printnumberedparparam \ocr \thenumber \text{\numprint{\thenumber}}}
\end{Verbatim}

The \numberedpar \thenum {⟨name⟩} opens numbered paragraph and prints it. The optional parameter is in \_the\_opt. You can re-define it if you need another design.

\numberedpar 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:

\begin{Verbatim}
\def\theorem {\medskip\bgroup\it \numberedpar A\{Theorem\}}
\def\endtheorem {\par\egroup\medskip}
\end{Verbatim}

\begin{Verbatim}
def\theorem \text{Let } M \text{ be... \endtheorem}
\end{Verbatim}

2.27 Lists, items
\_aboveliskip is used above the list of items, \_belowliskip is used below the list of items, \_setlistskip sets the skip dependent on the current level of items, \_listskipab is \_listskipamount or \_olistskipamount.

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 \beitems 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).

Various item marks are saved in \item:{letter} macros. You can re-define them or define more such macros. The \style{letter} does \_printitem=\{\item:{letter}\}. More exactly: \begitems does \_printitem=\_defaultitem first, then \style{letter} does \_printitem=\{\item:{letter}\} when it is used and finally, \_startitem alias * uses \_printitem.
\_sdef\{item:a\}{\_athe\{itemnum\}) }
\_sdef\{item:A\}{\_uppercase\{a\}\{itemnum\}) }
\_sdef\{item:x\}{\_raise.3ex\fullrectangle{.6ex}\_kern.4em}
\_sdef\{item:X\}{\_raise.2ex\fullrectangle{1ex}\_kern.5em}
\_sdef\{item:d\}{\_aftergroup\dword}
\_def\dword#1#2{{\bf #2 }\_ignorespaces} % #1 is \_ignorespaces from \_startitem
\_athe{\langle num \rangle} returns the \langle num \rangle s lowercase letter from the alphabet.
\fullrectangle{\langle dimen \rangle} prints full rectangle with given \langle dimen \rangle.
\_def\fullrectangle#1{\_hbox{\_vrule height#1 width#1}}
\_def\athe#1{\_ifcase#1\?\_or a\_or b\_or c\_or d\_or e\_or f\_or g\_or h\_or
\_or j\_or k\_or l\_or m\_or n\_or o\_or p\_or q\_or r\_or s\_or t\_or
\_or v\_or w\_or x\_or y\_or z\else \_fi}
\_public \begblock \endblock ;

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.
\_def\begblock{\_bgroup\_fnset \_medskip \_advance\leftskip by\_iindent \_firstnoindent}
\_def\endblock{\_par\_medskip\_egroup\_isnextchar\par{}\{\_noindent}}
\_public \begblock \endblock ;

2.28 Verbatim, listings
2.28.1 Inline and “display” verbatim
\_codedecl \begtt {Verbatim <2022-04-23>} % preloaded in format
The internal parameters \_ttskip, \_ttpenalty, \_viline, \_vifile and \_ttfont for verbatim macros are set.
\_def\_ttskip{\_medskip} % space above and below \begtt, \verbinput
\_mathchardef\_ttpenalty=100 % penalty between lines in \begtt, \verbinput
\_newcount\_viline % last line number in \verbinput
\_newread\_vifile % file given by \verinput
\_def\_ttfont{\_tt} % default tt font
\_protected\_sdef\{code\}#1{{\_escapechar=-1 \_ttfont \_the\everyintt \_relax
\_ea\_printinverbatim\_ea{\_detokenize{#1}}}}
\_def\_printinverbatim#1{\_leavevmode\_hbox{#1}}
\_regmacro {}{}{\_let\_code=\_detokenize \_let\_code=\_detokenize}
\_public \_code ;

The \begverbatim 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.
\verbchar {char} saves original catcode of previously declared \verbchar {char} (if such character was declared) using \verbchar \catcodechar{char} and \verbchar \savedttchar characters. Then new such values are stored. The declared character is activated by \verbchar \adef as a macro (active character) which opens a group, does \verbchar \setverb and other settings and reads its parameter until second the same character. This is done by the \verbchar \readverb macro. Finally, it prints scanned \texttt{<text>} by \verbchar \printinverbatim and closes group. Suppose that \verbchar \verbchar is used. Then the following work is schematically done:

\verbchar {\begingroup \setverb ... \readverb}
\verbchar {\readverb #1"{\printinverbatim{#1}\endgroup}\
\verbchar \verbchar}

Note that the second occurrence of \verbchar is not active because \verbchar 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 \begtti is run. It is defined by \eoldef, 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 \begtti macro does \setverb and another preprocessing, sets \endlinechar to \texttt{^J} and reads the following text in verbatim mode until \endtt occurs. This scanning is done by \startverb macro which is defined as:

\verbchar {\startverb #1\endtt #2^J{...}}

We must ensure that the backslash in \endtt has category 12 (this is a reason of the \texttt{\textbackslash} 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 redefine this macro.) The scanned data have ^J at each end of line and all spaces are active characters (defined as \texttt{\textbackslash}). Other characters have normal category 11 or 12.

The ^J is appended to verbatim data because we need to be sure that the data are finished by this character. When \endtt is preceded by spaces then we need to close these spaces by ^J and such line is not printed due to a trick used in \printverb.

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 \verbinput macro calls \verbinput{line} repeatedly to each scanned line of verbatim text. The \verbinput is used from \begtt...\endtt and from \verbinput too.

The \textcommentchars 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 \verbinput{line} is called here.

\verbinput{line} expects that it starts in vertical mode and it must do \par to return the vertical mode. The \verbinput{line} is used here: it does nothing when \ttline<0 else it prints the line number using \llap.

\puttpenalty puts \tppenalty before second and next lines, but not before first line in each \begtt...\endtt environment.

The \ttline is increased here in the \verbinput macro because of comments-blocks: the \verbinput is not processed in comments-blocks but we need to count the \ttline.

Macro \verbinput uses a file read previously or opens the given file. Then it runs the parameter scanning by \viscanparameter and \viscaminus. Finally the \verbinput is run. At the beginning of \verbinput, we have \viline= number of lines already read using previous \verbinput, \vnolines= 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 \vnolines 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 \verbinput loop.
\_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
\_def\_savemathsb{\_ifmathsb \_mathsboff \_def\_restoremathsb{\_mathsbon}\_fi}
\_def\_restoremathsb{}

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 to set the catcode to 12 (and space to 13). Examples:

\commentchars // % C++ comments
\commentchars -- % Lua comments
{\catcode`%=12 \_ea}\commentchars \%
{\catcode`#=12 \catcode` =13 \_ea}\commentchars#

There is one limitation when T_{\TeX} interprets the comments declared by \commentchars. Each block of comments is accumulated to one line and then it is re-interpreted by T_{\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. 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(first)(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 (\ifnum expands to \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.

The \visiblesp sets spaces as visible characters ␣. 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.
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:

\hisyntax-c.opm

OpTEX provides hisyntax-{c,lua,python,tex,html}.opm files. You can take inspiration from these files and declare more languages.
Users can re-declare default colors by \texttt{\_hicolors=\{list of color declarations\}}. This value has precedence over \texttt{\_hicolors\langle name\rangle} values declared in the \texttt{hicolors\langle name\rangle.omp} file. For example \texttt{\_hicolor=\{\texttt{\_tmpb}\texttt{\_Brown}\}} causes all strings in brown color.

Another way to set non-default colors is to declare \texttt{\newtoks\_hicolors\langle name\rangle} (without the \_ prefix) and set the color palette there. It has precedence before \texttt{\_hicolors\langle name\rangle} (with the \_ prefix) declared in the \texttt{hicolors\langle name\rangle.omp} file. You must re-declare all colors used in the corresponding \texttt{hisyntax\langle name\rangle.omp} file.

Notes for hi-syntax macro writers
The file \texttt{hisyntax\langle name\rangle.omp} is read only once and in a \TeX\ group. If there are definitions then they must be declared as global.

The file \texttt{hisyntax\langle name\rangle.omp} must (globally) declare \texttt{\_hisyntax\langle name\rangle} token list where the action over verbatim text is declared typically by using the \texttt{\replfromto} or \texttt{\replthis} macros.

The verbatim text is prepared by the \texttt{pre-processing phase}, then \texttt{\_hisyntax\langle name\rangle} is applied and then the \texttt{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 \texttt{\n\n}, so \texttt{\n\langle word\rangle\n} is the pattern for matching whole words (no subwords). The \texttt{\n} control sequence is removed in the post-processing phase.

• Each end of line is represented by \texttt{\n\~\n}.

• The \texttt{\_start} control sequence is added before the verbatim text and the \texttt{\_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.

• \texttt{\n} represents nothing but it should be used as a boundary of words as mentioned above.

• \texttt{\t} represents a tabulator. It is prepared as \texttt{\n\t\n} because it can be at the boundary word boundary.

• \texttt{\x \langle letter\rangle} \{\langle text\rangle\} can be used as replacing text. Consider the example

\texttt{\replfromto\{*\}{{*/}}{{/\#1*/}}}

This replaces all \texttt{C} comments /\*...*/ by \texttt{\x C\{/\#1\}/}. But \texttt{C} comments may span multiple lines, i.e. the \texttt{\~\~\~J} should be inside it.

The macro \texttt{\x \langle letter\rangle} \{\langle text\rangle\} is replaced by one or more occurrences of \texttt{\_z \langle letter\rangle} \{\langle text\rangle\} in the post-processing phase, each parameter \texttt{\langle text\rangle} of \texttt{\_z} is from from a single line. Parameters not crossing line boundary are represented by \texttt{\x C\{\langle text\rangle\}} and replaced by \texttt{\_z C\{\langle text\rangle\}} without any change. But:

\texttt{\x C\{\langle text1\}\~\~\_J\langle text2\}\~\~\_J\langle text3\}\}}

is replaced by

\texttt{\_z C\{\langle text1\}\~\_J\_z C\{\langle text2\}\~\_J\_z C\{\langle text3\}\}}

\texttt{\_z \langle letter\rangle} \{\langle text\rangle\} is expanded to \texttt{\_z:\langle letter\rangle} \{\langle text\rangle\} and if \texttt{\textcolor{color}{\_hicolor}} \langle letter\rangle \langle color\rangle is declared then \texttt{\_z:\langle letter\rangle} \{\langle text\rangle\} expands to \texttt{\textcolor{color}{\{color\}}\langle text\rangle}. So, required color is activated for each line separately (e.g. for \texttt{C} comments spanning multiple lines).

• \texttt{\y} \{\langle text\rangle\} is replaced by \texttt{\langle text\rangle} 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 \texttt{\_tmpb} macro.

\texttt{\replfromto \{\langle from\rangle\} \{\langle to\rangle\} \{\langle replacement\rangle\}} finds the first occurrence of \texttt{\langle from\rangle} and the first occurrence of \texttt{\langle to\rangle} following it. The \texttt{\langle text\rangle} between them is packed into \texttt{#1} and available to \texttt{\langle replacement\rangle} which ultimately replaces \texttt{\langle text\rangle}.

\texttt{\replfromto} continues by finding next \texttt{\langle from\rangle}, then, next \texttt{\langle to\rangle} repeatedly over the whole verbatim text. If the verbatim text ends with opening \texttt{\langle from\rangle} but has no closing \texttt{\langle to\rangle}, then \texttt{\langle to\rangle} 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.
\def\replfromto #1#2{\edef\tmpa{{#1}{#2}}\ea\replfromtoE\tmpa}
\def\replfromtoE#1#2#3{% #1=from #2=to #3=replacement
\def\replfrom##1#1##2{\addto\tmpb{##1}\
\ifx\fin##2\ea\replstop\else\afterfi{\replto##2}\fi}%
\def\replto##1#2##2{%\ifx\fin##2\afterfi{\replfin##1}\else\addto\tmpb{#3}\afterfi{\replfrom##2}\fi}%
\def\replfin##1#1\fin{\addto\tmpb{#3}\replstop}
\edef\tmpb{\ea}\ea\replfrom\tmpb#1\fin#2\fin\relax\replstop}
\def\replstop#1\fin\relax{}
\def\finrepl{}
\def\replthis#1#2{\edef\tmpa{{#1}{#2}}\ea\replstring\ea\tmpb \tmpa}
\public\replfromto \replthis ;
The \replthis {\langle pattern\rangle}{\langle replacement\rangle} replaces each \langle pattern\rangle by \langle replacement\rangle. Both parameters of \replthis are expanded first.
\def\replfromto/*}{*/}{\x C{/*#1/*}} replaces all C comments by \x C{...}. The patterns \langle from\rangle, \langle to\rangle and \langle pattern\rangle are not found when they are hidden in braces {...}. E.g. \replfromto{/*}{*/} replaces all C comments by \x C{...}. The patterns inside {...} are not used by next usage of \replfromto or \replthis macros.
\def\xscan#1#2^^J#3{\ifx\fin#3 \ea\xscanR\fi
\z{#1}{#2}%%
\ifx^#3\else ^^J\afterfi{\xscan{#1}#3}\fi}
\def\xscanR#1\fi#2^{^^J}
The \hicolor \langle letter \rangle \langle color \rangle defines \_z:\langle letter \rangle \langle text \rangle as \{\langle color \rangle\langle text \rangle\}. It should be used in the context of \x \langle letter \rangle \langle text \rangle macros.
\def\hicolor #1#2{\sdef{_z:#1}##1{{#2##1}}}
Aliases for languages can be declared like this. When \hsyntax{xml} is used then this is the same as \hsyntax{html}.

\def\hialias{xml}{html}
\def\hialias{json}{c}

2.29 Graphics

The \inspic is defined by \pdfximage and \pdrefximage primitives. If you want to use one picture more than once in your document, then the following code is recommended:

\newbox\mypic
\setbox\mypic = \hbox{\picw=3cm \inspic{⟨picture⟩}}

My picture: \copy\mypic, again my picture: \copy\mypic, etc.

This code downloads the picture data to the PFD output only once (when \setbox is processed). Each usage of \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 \setbox and all different sizes can be realized by the \transformbox{⟨transformation⟩}{\copy\mypic}.

Inkscape can save a picture to *.pdf file and labels for the picture to *.pdf_tex file. The second file is in \LaTeX{} format (unfortunately) and it is intended to read immediately after *.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 \inkinspic is used. These macros are stored in the \_inkdefs tokens list and it is used locally in the group. The solution is borrowed from OPmac trick 0032.
\_def\makeatletter\makeatother{}%
\_def\includegraphics[#1]#2{\_inkscanpage#1,page=,\_fin \_inspic\_the\_tmptoks\_hss}%
\_def\_inkscanpage#1page=#2,#3\_fin{\_ifx,#2,\_else\_picparams{page#2}\_fi}%
\_def\put(#1,#2)#3{\_nointerlineskip\_vbox to zo{\_vss\_hbox to zo{\_kern#1\_picwidth\_pdfsave\_hbox to zo{\_kern#3\_pdfrestore\_hss}}\_kern#2\_picwidth}}%
\_def\begin#1{\_csname _begin#1\_endcsname}%
\_def\_beginpicture(#1,#2){\_vbox\_bgroup\_hbox to\_picwidth{}\_kern#2\_picwidth \_def\end##1{\_egroup}}%
\_def\_begintabular[#1]#2#3\end#4{\_vtop{\_def\_\cr{\_cr}\_tabiteml{}\_tabitemr{}\_table{#2}{#3}}}%
\_def\color[#1]#2{\_scancolor #2,}%
\_def\_scancolor#1,#2,#3,{\_pdfliteral{#1 #2 #3 rg}}%
\_def\makebox(#1)[#2]#3{\_hbox to zo{\_csname _mbx:#2\_endcsname{#3}}}%
\_sdef{_mbx:lb}#1{#1\_hss}\_sdef{_mbx:rb}#1{\_hss#1}\_sdef{_mbx:b}#1{\_hss#1\_hss}%
\_sdef{_mbx:lt}#1{#1\_hss}\_sdef{_mbx:rt}#1{\_hss#1}\_sdef{_mbx:t}#1{\_hss#1\_hss}%
\_def\rotatebox#1#2{\_pdfrotate{#1}#2}%
\_def\lineheight#1{}%
\_def\setlength#1#2{}%
\_def\transparent#1{\_transparency\_exprA[0]{(1-#1)*255} }%
% Inkscape may generate \textbf{\textit{\textsc{TEXT}}}
\_def\textbf#1{\_begingroup\_let\_it\_bi\_bf #1\_endgroup}%
\_def\textit#1{\_begingroup\_it #1\_endgroup}%
\_def\textsl#1{\_begingroup\_trycs{slant}{}\_it #1\_endgroup}%
}
\public \inkinspic ;
\pdfscale{(x-scale)}{(y-scale)} and \pdfrotate{(degrees)} macros are implemented by \pdfsetmatrix primitive. We need to know the values of sin, cos function in the \pdfrotate. We use Lua code for this.
\_def\_pdfscale#1#2{\_pdfsetmatrix{#1 0 0 #2}}%
\_def\_gonfunc#1#2{\_directlua{tex.print(string.format(\_pcent.4f,math.#1(3.14159265*(#2)/180)))}}%
\_def\_sin{\_gonfunc{sin}}%
\_def\_cos{\_gonfunc{cos}}%
\_def\_pdfrotate#1{\_pdfsetmatrix{\_cos{#1} \_sin{#1} \_sin{(#1)-180} \_cos{#1}}}%
\_public \pdfscale \pdfrotate ;
The \transformbox{(transformation)}{(text)} is copied from OPmac trick 0046. The \rotbox{(degrees)}{(text)} is a combination of \rotsimple from OPmac trick 0101 and the \transformbox. Note, that \rotbox{-90} puts the rotated text to the height of the outer box (depth is zero) because code from \rotsimple is processed. But \rotbox{-90.0} puts the rotated text to the depth of the outer box (height is zero) because \transformbox is processed.
\_def\_multiplyMxV #1 #2 #3 #4 {% matrix * (vvalX, vvalY)
\_tmpdim = #1\_vvalX \_advance\_tmpdim by #3\_vvalY
\_vvalY = #4\_vvalY \_advance\_vvalY by #2\_vvalX
\_vvalX = \_tmpdim
\_def\_multiplyMxM #1 #2 #3 #4 {% currmatrix := currmatrix * matrix
\_vvalX=#1pt \_vvalY=#2pt \_ea\_multiplyMxV \_currmatrix
\_edef\_tmpb{\_ea\_ignorept\_the\_vvalX\_space \_ea\_ignorept\_the\_vvalY\_space \_ea\_ignorept\_the\_vvalX\_space \_ea\_ignorept\_the\_vvalY\_space \_ea\_ignorept\_the\_vvalX\_space \_ea\_ignorept\_the\_vvalY\_space \_ea\_ignorept\_the\_vvalX\_space \_ea\_ignorept\_the\_vvalY\_space}
\_def\_transformbox#1#2{\_hbox{\_setbox0=\_hbox{{#2}}\_dimendef\_vvalX 11 \_dimendef\_vvalY 12 \_use these variables \_dimendef\_newXt 13 \_dimendef\_newYp 14 \_only in this group \_dimendef\_newLt 15 \_dimendef\_newRp 16 \_prepare\transform{#1} \_kern\_newLt \_vrule height\_newXt depth\_newYp width\_zo \_netbox=\_bbox{\_box0}\_h=0\_zo \_dp=\_zo \_pdfsave#1\_rlap{\_box0}\_pdfrestore \_kern\_newRt}}%
\_def\_pretransform #1{\_def\_currmatrix{1 0 0 1 }% 
\_def\_pdfsetmatrix##1{\_edef\_tmpb{\_pdfextension setmatrix}% 
\_let\pdfsetmatrix=\_pdfsetmatrix #1% 
\_setnewHtDp 0pt \_ht0 \_setnewHtDp 0pt -\_dp0 
\_setnewHtDp \_wd0 \_ht0 \_setnewHtDp \_wd0 -\_dp0 
\_protected\_def \_pdfsetmatrix {
\_pdfsetmatrix\_pdfextension setmatrix}% 
\_let\pdfsetmatrix=\_pdfsetmatrix \_pdfextension setmatrix% 
\_def\_setnewHtDp #1 #2 {\_vvalX=#1\_relax \_vvalY=#2\_relax \_ea\_multiplyMxV \_currmatrix 
\_ifdim\_vvalX<\_newLt \_newLt=\_vvalX \_fi \_ifdim\_vvalX>\_newRt \_newRt=\_vvalX \_fi 
\_ifdim\_vvalY>\_newHt \_newHt=\_vvalY \_fi \_ifdim-\_vvalY>\_newDp \_newDp=-\_vvalY \_fi 
}\_def\_rotbox#1#2{\_isequal{90}{#1}\_iftrue \_rotboxA{#1}{\_kern\_ht0 \_tmpdim=\_dp0}{\_vfill}{#2}% 
\_else \_isequal{-90}{#1}\_iftrue \_rotboxA{#1}{\_kern\_dp0 \_tmpdim=\_ht0}{}{#2}% 
\_else \_transformbox{\_pdfrotate{#1}}{#2} \_fi \_fi 
}\_def\_rotboxA #1#2#3#4{\_hbox{\_setbox0=\_hbox{\_kern\_ht0 \_tmpdim=\_dp0}{\_vfill}{\_kern\_tmpdim}{#4}}% 
\_vbox to\_wd0{\_hbox to\_zo{\_kern\_tmpdim}{\_box0\_hss}}% 
\_kern\_tmpdim 
}\public \transformbox \rotbox ;

\_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{with}\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\text{x}\rangle}{\langle\text{y}\rangle} creates an ellipse with \langle\text{x}\rangle axis and \langle\text{y}\rangle axis. The origin is in the center. 
\_oval{\langle\text{x}\rangle}{\langle\text{y}\rangle}{\langle\text{roundness}\rangle} creates an oval with \langle\text{x}\rangle, \langle\text{y}\rangle size and with the given \langle\text{roundness}\rangle. The real size is bigger by 2\langle\text{roundness}\rangle. The origin is at the left bottom corner.
\mv{(x)}{(y)}{(curve)} moves current point to \(x\), \(y\), creates the \(curve\) and returns the current point back. All these macros are fully expandable and they can be used in the \pdfliteral argument.

\newdimen \lwidth
\def \fcolor{\let \fcolorvalue}
\def \lcolor{\let \lcolorvalue}
\def \shadow{\let \shadowvalue}
\def \overlapmargins{\let \overlapmarginsvalue}
\def \ratio{\isnextchar ={\ratioA}{\ratioA=}}
\def \ratioA="#1 {\def \ratiovalue{"1}}
\def \touppervalue#1{\ifx#1n \let #1=N \fi}
\def \setflcolors#1{% use only in a group
  \def \setcolor##1##2##3{##1 ##2}%
  \edef#1{\fcolorvalue}%
  \def \setcolor##1##2##3{##1 ##3}%
  \edef#1{#1\space\lcolorvalue\space}%
}
\optdef \inoval[\]{\vbox\bgroup
  \roundness=2pt \fcolor=Yellow \lcolor=Red \lwidth=.5bp
  \shadow=N \overlapmargins=N \hhkern=Opt \vvkern=Opt
  \the \ovalparams \relax \the \opt \relax
  \touppervalue \overlapmarginsvalue \touppervalue \shadowvalue
  \ifx \overlapmarginsvalue N\% \advance \hsize by-2\hhkern \advance \hsize by-2\roundness \fi
  \setbox0=\hbox\bgroup \kern \hhkern \let \next=%
}
\def \inovalA{\egroup % of \setbox0=\hbox\bgroup
  \ifdim \vvkern=\zo \else \ht0=\dimexpr \ht0+\vvkern \relax
    \dp0=\dimexpr \dp0+\vvkern \relax \fi
  \ifdim \hhkern=\zo \else \wd0=\dimexpr \wd0+\hhkern \relax \fi
  \ifx \overlapmarginsvalue N\% \dimen0=\roundness \dimen1=\roundness
    \else \dimen0=\hhkern \dimen1=\vvkern \fi
  \setcolors\tmp
  \bbox[\kerndimen0]
  \bbox to \zo[\kern\dimen0]\dp0
  \ifx \shadowvalue N\else
    \edef\tmpb{\bp{\wd0+\lwidth}\bp\ht0+\dp0+\lwidth}\bp{\roundness}%%
  \fi
}

The \inoval{\(text\)} is an example of \oval usage.
The \incircle{\(text\)} is an example of \circle usage.
The \ratio, \lwidth, \lcolor, \shadow and \overlapmargins are parameters, they can be set by user in optional brackets \[\ldots\]. For example \fcolor=\Red does \let \fcolorvalue=\Red and it means filling color.

The \setcolors uses the \setcolor macro to separate filling (non-stroking) color and stroking color. The \coc macro means "create oval or circle" and it expands to the stroking primitive \S or filling primitive \f or both \B. Only boundary stroking is performed after \fcolor=\relax. You cannot combine \fcolor=\relax with \shadow=Y.
Just before defining shadows, which require special graphics states, we define means for managing these
graphics states and other PDF page resources (graphics states, patterns, shadings, etc.). Our mechanism,
defined mostly in Lua (see 2.39.4, uses single dictionary for each PDF page resource type (extgstate, etc.)
for all pages (\pdfpageresources just points to it).

The macro \addextgstate{{PDF name}}{\{PDF dictionary\}} is a use of that general mechanism
and shall be used for adding more graphics states. It must be used after \dump. It’s general variant
defined in Lua is \_addpageresource {⟨resource type⟩}{⟨PDF name⟩}{⟨PDF dictionary⟩}. You can use
\pageresources or \_pageresources if you need to insert resource entries to manually created PDF
XObjects.

\public \inoval \incircle \ratio \lwidth \fcolor \lcolor \shadow \overlapmargins;

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\{\(\text{curve}\)\} does the shadow effect.

A generic macro \_clipinpath\{\(x\) \(y\) \(\text{curve}\) \(\text{text}\)\} declares a clipping path by the \(\text{curve}\) shifted by the \(\langle x \rangle\), \(\langle y \rangle\). The \(\langle \text{text} \rangle\) is typeset when such clipping path is active. Dimensions are given by bp without the unit here. The macros \_clipinoval\{\(x\) \(y\) \(\text{width}\) \(\text{height}\) \{\(\text{text}\)\} and \_clipincircle\{\(x\) \(y\) \(\text{width}\) \(\text{height}\) \{\(\text{text}\)\} are defined here. These macros read normal \TeX\ dimensions in their parameters.
2.30 The \table macro, tables and rules

2.30.1 The boundary declarator:
The \{\textlangle declaration\}\{\textlangle data\}\} includes column declarators (letters) and other material: the | or \{(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 \{l|lc(xx)(yy)c\} should be written more exactly using the boundary declarator : by \{l|lc(xx)(yy):c\}. But you can set these boundaries to other places using the boundary declarator : explicitly, for example \{lc|l|lc(xx)(yy)c\}. The boundary declarator : can be used only once between each pair of column declarators.

Each table item has its group. The \langle cmd\rangle 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 \langle setting\rangle followed by column declarator. But if the column is not first, you must use :\langle setting\rangle. 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}{\langle data\rangle}

2.30.2 Usage of the \tabskip primitive
The value of \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, \tabskip is 0pt. It means that only \textlangle tabiteml, \textlangle tabitemr and \langle cmdss\rangle 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 \tabskip value declared before the \table macro (or in \teverytable or in \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 \langle \tabskip=value\rangle in the \textlangle declaration\} immediately before boundary character. The boundary character represents the column pair for which the \tabskip has individual value. For example \textlangle c(\tabskip=5pt):r\rangle gives \tabskip value between \textlangle c\rangle and \textlangle r\rangle columns. You need not use boundary character explicitly, so \textlangle c\tabskip=5pt)r\rangle gives the same result.

Space before the first column is given by the \tabskipl and space after the last column is equal to \tabskipr. Default values are 0pt.

Use nonzero \tabskip only in special applications. If \tabskip is nonzero then horizontal lines generated by \crli, \crlli and \crlp have another behavior than you probably expected: they are interrupted in each \tabskip space.

2.30.3 Tables to given width
There are two possibilities how to create tables to given width:

- \table to\langle size\}\{\textlangle declaration\}\{\textlangle data\}\} uses stretchability or shrinkability of all spaces between columns generated by \tabskip value and eventually by \textlangle \tabskipl, \tabskipr\} values. See example below.
- \table pxto\langle size\}\{\textlangle declaration\}\{\textlangle data\}\} expands the columns declared by \langle size\} if the \langle size\} is given by a virtual \textlangle \tsize\} unit. See the example below.

Example of \textlangle table to\langle size\}:\thistable{\textlangle \tabskip=0pt plus1fil minus1fil\}\table to\langle hsize\}\{lr\}\{\textlangle data\}\}

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 \textlangle table pxto\langle size\} (means “paragraphs expanded to”):
\table pxto\langle hsize \}\{lc\{\textlangle \tsize\}\}\{\textlangle data\}\)
\hsize
aaa  & Ddkas jd dsjds ds cgha sfgs dd fddzf dfhz xxz
     & dras ffg hkad kds d sdjds h sd jd dsjds ds cgha
     & sfgs dd fddzf dfhz xxz. \crl
bb ddd ggg  & Dsjds ds cgha sfgs dd fddzf dfhz xxz
     & ddkas jd dsjds ds cgha sfgs dd fddzf. \crl
The first column is variable width (it gets the width of the most wide item) and the resting space to given \hspace is filled by the p column.

You can declare more than one p{⟨coefficient\hspace⟩} columns in the table when pxto keyword is used.

\table pxto13cm {r p{3.5\hspace} p{2\hspace} p{\hspace} l}{{{\hbox{data}}}{{\hbox{data}}}{{\hbox{data}}}}
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 \{⟨label⟩\{⟨text⟩\} in the first run of \TeX. But the widths of all boxes with the same label are saved to \ref file and the maximum box width for each label is calculated at the beginning of the next \TeX run. Then \eqbox \{⟨label⟩\{⟨text⟩\} behaves like \hbox to \langledim:label⟩ \{hss \langletext⟩\hss\}, where \langledim:label⟩ is the maximum width of all boxes labeled by the same \{⟨label⟩\}. The documentation of the \LaTeX package eqparbox includes more information and tips.

The \eqboxsize \{⟨label⟩\} expands to \langledim:label⟩ if this value is known, else it expands to the given \langledim⟩.

The optional parameter r or l can be written before \{⟨label⟩\} (for example \eqbox r[⟨label⟩]\{⟨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.

\def\leftitem#1{\par
  \noindent \hangindent=\eqboxsize[items]2em \hangafter=1
  \eqbox r[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{⟨declaration⟩}\{⟨data⟩\} macro is inserted into \_tablebox. You can change default value if you want by \let\_tablebox=\vtop or \let\_tablebox=\relax.

\def\let\_tablebox=\_vbox
We save the to(size) or pxto(size) to \#1 and \_tableW sets the to(size) to the \_tableW macro. If pxto(size) is used then \_tableW is empty and \_tmpdim includes given \langle size⟩. The \_ifpxto returns true in this case.

The \table continues by reading \{⟨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.

\_newifi \_ifpxto
\_def\_table#1#{\_tablebox\_bgroup \_tableW#1\_empty\_fin
\_bgroup \catcodetable \_optexcatcodes \_tableA}
\_def\_tableW#1#2\_fin{\_pxtofalse
\_ifx#1\_empty \_def\_tablew{}\_else
\_ifx#1p \_def\_tablew{}\_tableWx#2\_fin \_else \_def\_tablew{#1#2}\_fi\_fi
\_def\_tableWx xto#1\_fin{\_tmpdim=#1\_relax \_pxtotrue}
\_public \_table ;

The \tablinespace is implemented by enlarging given \tabstrut by desired dimension (height and depth too) and by setting \_lineskip=-2\_tablinespace. Normal table rows (where no \hrule is between them) have normal baseline distance.

155
The \_tableA\{\langle declaration \rangle\} macro scans the \{\langle declaration \rangle\} by \_scantabdata\#1\_{\_relax} and continues by processing \{\langle data \rangle\} by \_tableB. The trick \_tmptoks=\{\langle data \rangle\}\_edef\_tmpb{\_the\_tmptoks} is used here in order to keep the hash marks in the \{\langle data \rangle\} unchanged.

The \_tableB saves \{\langle data \rangle\} to \_tmpb and does \replstrings to prefix each macro \crl (etc.) by \_crcr. See \_tabreplstrings. The whole \_tableB macro is hidden in \{\ldots\} in order to there may be \table in \table and we want to manipulate with \& and \cr as with normal tokens in the \_tabreplstrings, not as the item delimiters of an outer \table.

The \tabskip value is saved for places between columns into the \_tabskipmid macro. Then it runs

\tabskip=\tabskipl \halign{\langle converted declaration \rangle} \tabskip=\tabskipr \cr \langle data \rangle\crcr}

This sets the desired boundary values of \tabskip. The “between-columns” values are set as \tabskip=\_tabskipmid in the \{converted declaration\} immediately after each column declarator.

If pxto keyword was used, then we set the virtual unit \tsize to \_hsize first. Then the first attempt of the table is created in box 0. All columns where p{\ldots\tsize} is used, are created as empty in this first pass. So, the \wd0 is the width of all other columns. The \tsizesum includes the sum of \tsize’s in \hsize units after first pass. The desired table width is stored in the \_tmpdim, so \_tmpdim-\wd0 is the rest which have to be filled by \tsizes. Then the \tsize is re-calculated and the real table is printed by \halign.

If no pxto keyword was used, then we print the table using \halign directly. The \_tablew macro is nonempty if the to keyword was used.

The \{\langle data \rangle\} are re-tokenized by \_scantextokens in order to be more robust to catcode changing inside the \{\langle data \rangle\}. But inline verbatim cannot work in special cases here like `{` for example.

\_tabreplstrings 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 \texttt{\scantabdata} macro converts \texttt{table}'s \langle declaration \rangle to \texttt{\halign} \langle converted declaration \rangle. The result is stored into \texttt{\tabdata} tokens list. For example, the following result is generated when \langle declaration \rangle=|cr||cl|.

\begin{verbatim}
  tabdata: \_vrule\_the\_tabiteml\{\_hfil\#\_unsskip\_hfil\}\_the\_tabitemr\_tabstrutA
               \_vrule\_kern\_vvkern\_vrule\_tabstrutA
               \_the\_tabiteml\{\_hfil\#\_unsskip\_hfil\}\_the\_tabitemr\_tabstrutA

ddlinedata: &\_dditem &\_dditem\_vvitem &\_dditem &\_dditem
\end{verbatim}

The second result in the \texttt{\ddlinedata} macro is a template of one row of the table used by \texttt{\crli} macro.

The \texttt{\addtabitemx} adds the boundary code (used between columns) to the \langle converted declaration \rangle. This code is \texttt{\\the\colnum=} \langle value \rangle \relax. You can get the current number of column from the \texttt{\colnum} register, but you cannot write \texttt{\the\colnum} as the first object in a \langle data \rangle item because \texttt{\halign} first expands the front of the item and the left part of the column declaration is processed after this. Use \texttt{\relax\the\colnum} instead. Or you can write:

\begin{verbatim}
  \def\showcolnum{\ea\def\ea{totcolnum}\ea\the\colnum}\the\colnum/\totcolnum}
  \table{ccc}{\showcolnum & \showcolnum & \showcolnum}
\end{verbatim}

This example prints 1/3 2/3 3/3, because the value of the \texttt{\colnum} is equal to the total number of columns before left part of the column declaration is processed.
This code converts || or | from \table ⟨declaration⟩ to the ⟨converted declaration⟩.

The default “declaration letters” c, 1, 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⟩{...} 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:

\table{|c|c|}{\crl \Red A & B \crl}

We don’t want vertical line after red A to be in red.

The \paramtabdeclare{⟨data⟩} is invoked when p{⟨data⟩} declarator is used. First, it saves the \hsize 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 \hsize 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.

\tableparB initializes the paragraphs inside the table item and \tableparC closes them. They are used in the \paramtabdeclare macro. The first paragraph is no indented.
This macro isn’t only the primitive \unskip because we allow usage of plain \TeX \hideskip macro: &\hideskip text\hideskip&.

The \texttt{\textbackslash fL}, \texttt{\textbackslash fR}, \texttt{\textbackslash fC} and \texttt{\textbackslash fX} macros only do special parameters settings for paragraph building algorithm.

The \texttt{\textbackslash S} macro is more tricky. The \texttt{\textbackslash tableparbox} isn’t printed immediately, but \texttt{\textbackslash setbox2=} is prefixed by the macro \texttt{\textbackslash tableparA}, which is empty by default (used in \texttt{\textbackslash tablepar}). The \texttt{\textbackslash tableparD} is processed after the box is set: it checks if there is only one line and prints \texttt{\hbox to\hsize{\hfil ⟨this line⟩ \hfil}} in this case. In other cases, the box\texttt{2} is printed.

The family of \texttt{\textbackslash cr*} macros \texttt{\textbackslash cr}, \texttt{\textbackslash crll}, \texttt{\textbackslash crli}, \texttt{\textbackslash crlli}, \texttt{\textbackslash crlp} and \texttt{\textbackslash tskip ⟨dimen⟩} is implemented here. The \texttt{\textbackslash zerotabrule} is used to suppress the negative \texttt{\lineskip} declared by \texttt{\textbackslash tablinespace}.

The \texttt{\textbackslash mspan⟨number⟩[⟨declaration⟩]{⟨text⟩}} macro generates similar \texttt{\omit \span\omit\span} sequence as plain \TeX macro \texttt{\multispan}. Moreover, it uses \texttt{\scantabdata} to convert \texttt{⟨declaration⟩} from \texttt{\table} syntax to \texttt{\halign} syntax.
The \texttt{\textbackslash vspan\{number\}\{text\}} implementation is here. We need to lower the box by 
\((\text{number} - 1) \ast (\ht + \dp \text{ of } \tabstrut) / 2\).

The \#1 parameter must be a one-digit number. If you want to set more digits then use braces.

The parameters of primitive \texttt{\textbackslash vrule} and \texttt{\textbackslash hrule} keeps the rule “last wins”. If we re-define \texttt{\textbackslash hrule} to \texttt{\orihrule height1pt} then each usage of redefined \texttt{\hrule} uses 1pt height if this parameter isn’t overwritten by another following \texttt{\hrule parameter}. This principle is used for settings another default rule thickness than 0.4pt by the macro \texttt{\rulewidth}.

The \texttt{\frame\{text\}} uses “\texttt{\vbox in \vtop}” trick in order to keep the baseline of the internal text at the same level as outer baseline. User can write \texttt{\frame\{abcxyz\}} in normal paragraph line, for example and gets the expected result: \texttt{abcxyz}. The internal margins are set by \texttt{\vvkern} and \texttt{\hhkern} parameters.

\texttt{\eqbox} and \texttt{\eqboxsize} are implemented here. The widths of all \texttt{\eqbox}es are saved to the .ref file in the format \texttt{\_Xeqbox\{\langle label \rangle\}\{\langle size \rangle\}}. The .ref file is read again and maximum box width for each \langle label \rangle is saved to \_eqb:\langle label \rangle.

\subsection{2.31 Balanced multi-columns}

\texttt{\betweencolumns} or \texttt{\leftofcolumns} or \texttt{\rightofcolumns} include a material printed between columns or left of all columns or right of all columns respectively. The \texttt{\betweencolumns} must include a stretchability or a material with exactly \texttt{\colsep width}. You can redefine these macros. For example
the rule between columns can be reached by `\_def\_betweencolumns{\hss\vrule\hss}`. `\_multiskip` puts its material at the start and at the end of `\begmulti...\endmulti`. The code used here 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`. Then the macro measures the amount of free space at the current page using `\pagegoal` and `\pagetotal` and does `\vsplit` of `\vbox` to columns with a height of such free space. This is done only if we have enough amount of material in `\vbox` to fill the full page by columns. This is repeated in a loop until we have less amount of material in `\vbox`. 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 `\vbox{\{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).

Splitting columns...
Final balancing of the columns.

\_balancecolumns \_ifdim \_dimen0>\_dimen2 \_ea \_ea \_ea \_splitpart
\_else \_balancecolumns \_fi
\fi
\}

\_setcolsize \_dimen2=\_dimen1
\_advance \_dimen2 by-\_baselineskip
%% split the material to more pages?
\_ifvoid6 \_else
\_ifdim \_dimen0>\_dimen2 \_ea \_ea \_ea \_splitpart
\_else \_balancecolumns \_last balancing
\_fi \_fi
\}

2.32 Citations, bibliography

2.32.1 Macros for citations and bibliography preloaded in the format

\_newcount \_bibnum % the bibitem counter
\_newtoks \_bibmark % the bibmark used if \nonumcitations
\_newcount \_lastcitenum \_lastcitenum=0 % for \shortcitations
\_public \\bibnum \\bibmark ;
\_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\}.

\_def \_bibp(\_the \_bibpart/)% unique name for each bibliography list

\cite{[label], [label], ...} manages \_cite\_A and prints \{bib-marks\} using \_printsavedcites.

\nocite{[label], [label], ...} only manages \{labels\} but prints nothing.

\rcite{[label], [label], ...} behaves like \cite but prints \{bib-marks\} without brackets.

\ecite{[text]} behaves like \rcite{[label]} but prints \{text\} instead \{bib-mark\}. The \{text\} is hyperlinked like \{bib-marks\} when \cite or \rcite is used.

Each \cite/\rcite/\ecite macro starts from empty list of \{bib-marks\} because new group is opened.

\label defines \_Xbib macros. It adds the \{label\} to a global list \_ctlst:\{bibpart\}/ 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 say space and to avoid duplications of \{label\} in the \_ctlst:\{bibpart\}/, we distinguish four cases:

- \{label\} was not declared by \_Xbib before and it is first such a \{label\} in the document: Then \bib:\{bibpart\}/\{label\} is undefined and we save label using \_addcitelist, write warning on the terminal and define \bib:\{bibpart\}/\{label\} as empty.
- \{label\} was not declared by \_Xbib before but it was used previously in the document: Then \bib:\{bibpart\}/\{label\} is empty and we do nothing (only data to \_savedcites are saved).
- \{label\} was declared by \_Xbib before and it is first such \{label\} used in the document: Then \bib:\{bibpart\}/\{label\} includes \bib\{number\} and we test this case by the command \if \& \bib\{number\}. This is true when \bib\{number\} expands to empty. The \{label\} is saved by \_addcitelist and \bib:\{bibpart\}/\{label\} is re-defined directly as \{number\}.
- \{label\} was declared by \_Xbib and it was used previously in the document. Then we do nothing (only data to \_savedcites are saved).

The \_cite\_A macro runs repeatedly over the whole list of \{labels\}.

\bibcite accepts \{bibpart\}/\{label\} to a global list \_ctlst:\{bibpart\}/ 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).

\citebib omits \{bibpart\}/\{label\} 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).

\citetbib accepts \{bibpart\}/\{label\} to a global list \_ctlst:\{bibpart\}/ 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 implement possibility of more independent bibliography lists distinguished by \texttt{\bibpart}, the \texttt{\addcitelist{\texttt{\label}}} macro must add the \texttt{\label} to given \texttt{\ctlst}:

\begin{verbatim}
\def\addcitelist#1{\unless \ifcsname \ctlst:\bibp\endcsname \sxdef{\ctlst:\bibp}{} \fi \ifx \csname \ctlst:\bibp\endcsname \write \openref \immediate \wref \Xcite{\bibp}{#1}\fi \else \global \ea \addto \csname \ctlst:\bibp\endcsname \citeI[#1]\fi}
\end{verbatim}

When \texttt{\addcitelist} is processed before \texttt{\usebib}, then \texttt{\citeI[\texttt{\label}] is added.} \texttt{\usebib} will use this list for selecting right records from .bib file. Then \texttt{\usebib} sets \texttt{\ctlst:\bibp} to \texttt{\write}. If \texttt{\addcitelist} is processed after \texttt{\usebib}, then \texttt{\Xcite{\bibpart}{\texttt{\label}}} is saved to the .ref file. The \texttt{\Xcite} creates \texttt{\ctlstB:\bibpart} as a list of saved \texttt{\citeI[\texttt{\label}]}. Finally, \texttt{\usebib} concats both lists \texttt{\ctlst:\bibpart} and \texttt{\ctlstB:\bibpart} in the second \TeX run.

The \texttt{\bibmarks} (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{\shortcitations} 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{\shortcitations} sets simply \texttt{\lastcitenum=1}. The macros for \texttt{\bibmarks} printing follows (sorry, without detail documentation). They are documented in \texttt{opmac-d.pdf} (but only in Czech).
The \bib[(\label)] or \bib[(\label)]=\{\bibmark\} prints one bib-entry without reading any database. The bib-entry follows after this command. This command counts the used \bib from one by \bibnum counter and saves \Xbib{\bibpart}{\label}{\number}{\nonumber} into .ref file immediately using \wbib{\label}{\number}{\nonumber}. This is the core of creation of mapping from \labels to \number and \nonumber.

\bibA and \bibB implement the scanner of the optional argument with the \bibmark.

\bibgl is \relax by default but \slides do \let\bibgl=\global. \dbib[(\label)] 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 \bibmarks by \llap{\the\bibnum} or \the\bibnum. If \nonumcitations then the \citelinkA is not empty and \bibmarks \the\bibnum nor \the\bibmark 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.
The \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.txt 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.

\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 document). 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 \doctype{BOOK}, \doctype{ARTICLE}) and declares their mandatory and optional fields (like \field{author}, \field{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 \{\texto}. Recommendation: convert such old files to Unicode encoding. If you are unable to do this then you can set \bibtexthook={\oldaccents}.

2.32.3 Notes for bib-style writers

The .bib files include records in the format:

\begin{verbatim}
@\entry-type{\{label\},
  \field-name = "\{field-data\}",
  \field-name = "\{field-data\}",
  ...etc
}
\end{verbatim}

see the file demo/op-biblist.bib for a real example. The \entry-types and \field-names are case insensitive. More field-names can behave equally if the \fieldalias{\new-field-name}\{\given-field-name\} is used in a style file. If a \new-field-name is declared by this command and it is used in the .bib file then the effect is the same as if it was used the \given-field-name.

Ancient Bib\TeX\ has read such files and has generated files appropriate for reading by \LaTeX. It has worked with a set of \entry-types, see the www page \url{http://en.wikipedia.org/wiki/Bib\TeX}. 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 2.32.6.
The printing rules for each \{entry-type\} must be declared by \sdef{\_print:(entry-type)} in \{style\}.opm file. The \{entry-type\} has to be lowercase here. \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.
- \_bprintfb \{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 \_bprintfb commands are nested (\_bprintfb in \_bprintb), then you need to write the ####1 parameter for internal \_bprintfb. 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 file 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 \TeX{} (and may be redefined by the author or document designer).

The \_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 precessed 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
- \_namecount: 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 \text{\_{mt: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 2.37.2 for more detail. Phrases for \{identifiers\}: and, etal, edition, citedate, volume, number, prepages, postpages, editor, editors, available, availablealso, backthesis, masthesis, phdthesis are defined already, see the end of section 2.37.2.

The \_preparebibsorting field is declared by \_readbibs as a special field where sorting phrase can be specified. If it is present then it has precedence before default sorting phrase generated by \_preparebibsorting from the lastname, firstnames of the first author and from the year. Suppose that the .bib file includes:

\begin{verbatim}
author    = "Jan Chadima",
sortedby  = "Hzzadima Jan",
\end{verbatim}

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 \text{sortedby = "0"}, because this character is sorted before A.

If you want to declare a different sorting rule, you can re-define the \_preparebibsorting macro. The example is in the \TeX{} trick 0113.
2.32.4 Direct reading of \texttt{.bib} files

\texttt{\readbibs \{\texttt{\{bib-bases\}}\}} is internally used (by \texttt{\usebib}) for reading \texttt{.bib} databases in Bib\TeX\ format. The \texttt{\{bib-bases\}} is comma separated list of file names (without \texttt{.bib} extension, without spaces). These files are read and \texttt{\readbibs} defines macros \texttt{\_be:/(bibpart)/\{label\}}, where \texttt{\{label\}} is the label of the reference record. These macros include key-value pairs \{\{field name\}\}\{\{field data\}\}. The first pair is \{\{\texttt{entry type}\}\}. For example, if we have in the \texttt{.bib} file:

\begin{verbatim}
@Book { tbn,  
  author = "Petr Olšák",  
  TITLE = {\TeX{}book naruby},  
  publisher = "Konvoj",  
  year = 2001, 
}
\end{verbatim}

and the \texttt{\bibpart} is empty (default value) then the \texttt{\_be/tbn} macro is defined with the content:

\begin{verbatim}
[0]{BOOK}[author]{{Olšák}{Petr}{}}[authornum]{1}[title]{{\TeX{}book naruby}}%  
[publisher]{Konvoj}[year]{2001}
\end{verbatim}

If you do \texttt{\slet{tmp}{\_be:/tbn}} then you can print the data (for example) by:

\begin{verbatim}
\ea\foreach \tmp \do [#1]#2{\wterm{field-name: "#1", data: "#2"}}
\end{verbatim}

or you can do \texttt{\ea\foreach \tmp \do [#1]#2{\sdef{bib-field:#1}{#2}}} to enable direct access to the scanned data.

Note that entry type and field names are converted lower-case by the \texttt{\readbibs} macro.

There are two special entry types: \texttt{\COMMENT{\{ignored text\}}} and \texttt{\TEXCODE{\{processed text\}}}. The \texttt{\COMMENT{\{\}}} is ignored, the \texttt{\TEXCODE{\{\}}} is executed by \TeX{}. The definitions of macros used in other entries in data of fields can be here. If the \texttt{\usebib} is used then the \texttt{\TEXCODE{\{\}}} is executed inside a \TeX{} group, so the assignment is locally valid only during creating the reference list. The BiB\TeX{}'s \texttt{\STRING{\{\}}} is not supported. All others entry types are interpreted as a \texttt{reference entry} and they are interpreted as described above. An optional balanced text between entries in \texttt{.bib} files is ignored.

If the macro \texttt{\_be:/(bibpart)/\*} is defined then the \texttt{\readbibs} macro reads all entries from \texttt{.bib} files and creates \texttt{\_citelist}. If the \texttt{\_be:/(bibpart)/\*} is undefined then the \texttt{\readbibs} macro reads only entries with \texttt{\_be:/(bibpart)/\{label\}} set to the empty macro. After reading, the macros \texttt{\_be:/(bibpart)/\{label\}} are globally re-defined as described above.

The \texttt{\readbibs} macro doesn't convert fields data, but there are two exceptions: author and editor fields. These fields have very specific format with various alternatives, see \url{https://nwalsh.com/tex/teXhelp/bibtx-23.html}. Shortly speaking, more authors are divided by the \texttt{and} keyword and names of a single author must be separated to four subfields: \texttt{\{Lastnames\}}, \texttt{\{Firstnames\}}, \texttt{\{Von\}}, \texttt{\{Junior\}}. Only the \texttt{\{Lastnames\}} subfield must be nonempty. The input can look like

\begin{verbatim}
Leonardo Piero da Vinci
or
da Vinci, Leonardo Piero
\end{verbatim}

and both these variants are converted to \texttt{\{Vinci\}{\{Leonardo Piero\}{da}\{\}}}{}. The \texttt{\{Von\}} part is recognized as a word with only lowercase characters. In general, the name can be written without commas: \texttt{\{Firstnames\} \{Von\} \{Lastnames\}} or with single comma: \texttt{\{Von\} \{Lastnames\}}, \texttt{\{Firstnames\}} or with two commas: \texttt{\{Von\} \{Lastnames\}, \{Junior\}, \{Firstnames\}} and all these variants are converted to the quaternion \texttt{\{\{Lastnames\}\{\{Firstnames\}\}\{\{Von\}\}\{\{Junior\}\}}} by the \texttt{\readbibs} macro. If there are more than single author, then each author is saved in four subfields side by side, so you have 4 or 8 or 12 etc. subfields in the author/editor data field. You can read them by \texttt{\foreach \{author-data\} \do [#1]#2#3#4{\ldots}}.

2.32.5 The \texttt{\usebib\textunderscore opm} macro file loaded when \texttt{\usebib} is used

\texttt{\\_codedecl \readbibs \{Reading bib databases <2023-06-25>\} \% loaded on demand by \usebib \usebib\textunderscore opm}

First, we implement the scanner of \texttt{.bib} files. Unfortunately, the format of these files isn't \TeX{} friendly, so we must do more work. \texttt{\readbibs \{\texttt{\{bib-bases\}}\}} reads \texttt{\{bib-bases\}} files (i.e. the BiB\TeX{} format).
The \_nextat macro skips the text in the .bib file to the next @, and starts the \bibentry macro which reads @\{entry type\}@\{data\} from the .bib file. Each reference entry is converted to the \_entrydata macro and then \glet \_be:\{bib-part\}/\label = \_entrydata is done. The \_entrydata includes key-value pairs, as described in the section 2.32.4.

\nextfield reads next field name and saves it to the \_fieldname and then reads field data and saves it to the \_fielddata.
The \_auscan\{authors/editors-names\}\{field-name\} reads the specific BibTeX format mentioned in section 2.32.4 and converts them to \{Lastname\}\{Firstname\}\{Von\}\{Junior\} for each author/editor. The result includes 4 \_k\ subfields (where \_k\ is number of the authors/editors) and it is saved to the \_entrydata\ and the [authornum]\{\_k\} or [editornum]\{\_k\} is added.

The \_auscanA\ macro does the loop over authors separated by and. Each single author has its \_tmpb\ macro with \_x\ and \_x. Each letter corresponds to single word of the name (\_X\: begins with uppercase, \_x: begins with lowercase). For example Leonardo Piero da Vinci has \_tmpb\ macro XXXX.. If there are commas in after some words, then these commas are in \_tmpb\ macro too, for example da Vinci, Piero Leonardo has its \_tmpb\ macro xX,XX.. The number of commas is saved to \_tmpnum. The \_auscanB\ macro does a slight modifications of the \_tmpb\ macro as mentioned in comments. Then the macro \_auscanD\{\_tmpb-pattern\}\{\_WordA\}\{\_WordB\}\{\_WordC\}... is executed. It saves given words due to the \_tmpb\ pattern to the macros \_Lastname, \_Firstname, \_Von, \_Junior in a loop. Finally, the contents of these macros are saved to \_fielddata\ and then to the \_entrydata.

\_fieldalias\{(new-name)\}(given-name) defines \_fia:\{new-name\} as \{given-name\}.
The _citelist includes \citeI{[label]} commands. The \usebib macro runs this lists in order to print references. Each \citeI{[label]} prints single bib entry given by the [label]. It opens a group, sets macros \fd{[field-name]} to \field{[field-data]} and runs \printentry. Finally, it closes \TeX group, so all macros \fd{[field-name]} have their initial (undefined) value.

The \getfield{[field-name]} \macro does \defmacro{[field-data]}. If the field isn’t declared then the \macro is empty.

\preparebibsorting is called repeatedly for each bib entry when its reading from .bib file is finished. Its main goal is to do \gdef{bes:{citekey}} {\{[sorting-rule]\}~~~{citekey}}. Note that the part of the control sequence name after \~~~ is ignored during sorting. The default \preparebibsorting macro creates \{sorting-rule\} in the form: \(\langle\text{Lastnames}\rangle\ \langle\text{Firstnames}\rangle\ \langle\text{Von}\rangle\ \langle\text{Junior}\rangle\) of the first author followed by \langle\text{year}\rangle from \text{year} field.

\dobbisorting\_citelist sorts the \_citelist and runs it.
The \_printentry macro prints bibliographic reference entry. It prints \textit{(bibnum)} or \textit{(bimark)} (including hyperlinks) and they are followed by printing the entry data. The format is given by the \_printbib macro and by \_print:\textit{(entrytype)} declared in the bib-style file.

The \_bprinta, \_bprintb, \_bprintc, \_bprintv commands used in the style files:
\_loopauthors\{(field-name)\} does a loop over all authors/editors in the author or editor field. The \_namecount (total number of authors/editors) was defined in \_bprintb. Then for each author/editor it does:

- Set \_NameCount to the position number of the currently processed author/editor.
- Define \_Lastname, \_Firstname, \_Junior, \_Von, \_After macros.
- Run \_authorname or \_editorname macro (defined in the bib style file).

\_bibwarning can be used if the mandatory field is missing. Note that \nobibwarnlist is used here, it is set by \nobibwarning macro.

The \usebib command is defined as \input{usebib.opm}\_usebib in the format. So, the command is re-defined here and it is run again with the new meaning. The \usebib macro defined here reads \_ctlst:\{bibpart\}/ and \_ctlstB:\{bibpart\}/ (they include a list of \_citeI\{⟨label⟩\}) and merges them to a single \_citelist. Then the style file is read in a group, the \readbibs macro reads given .bib files and resulting \_citelist is processed: i.e. the macros \_citeI print desired entries.
2.32.6 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:

```
\fontfam[LMfonts]
\nocite[*]
\usebib/s (iso690) op-biblist
\end
```

Common rules in .bib files

There are entries of type @FOO{...} in the .bib file. Each entry consists of fields in the form \texttt{name}_{m}="value", or \texttt{name}_{m}={value}. No matter which form is used. If the value is pure numeric then you can say simply \texttt{name}_{m}=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 not 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). 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:

- \texttt{lang} = language of the entry. The hyphenation plus autogenerated phrases and abbreviations will be typeset by this language.
- \texttt{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 \texttt{\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 \texttt{von} part is typically missing):

\begin{itemize}
  \item Firstname(s) von Lastname
  \item von Lastname, Firstname(s)
\end{itemize}
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 $\text{\caps\_rm\space Lastname}$.

You can specify the option $\text{aumax:\langle number\rangle}$: The $\langle number\rangle$ denotes the maximum authors to be printed. The rest of the authors are ignored and the \textit{et\-al.} is appended to the list of printed authors. This text is printed only if the \texttt{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 \textit{et\-al.}, then you can use \texttt{auetal} option.

There is an $\text{aumin:\langle number\rangle}$ option which denotes the definitive number of printed authors if the author list is not fully printed due to \texttt{aumax}. If \texttt{aumin} is unused then \texttt{aumax} authors are printed in this case.

All authors are printed if \texttt{aumax:\langle number\rangle} option isn’t given. There is no internal limit. But you can set the global options in your document by setting the \texttt{\biboptions} tokens list. For example:

\begin{verbatim}
\biboptions={aumax:7 aumin:1}
\% if there are 8 or more authors then only the first author is printed.
\end{verbatim}

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 \texttt{\auprint:\{value\}} option. The \{value\} will be printed instead of the authors list. The \{value\} can include \texttt{\textbackslash AU} macro which expands to the authors list. Example:

\begin{verbatim}
author = "Robert Calbraith",
option = "auprint:\{\textbackslash AU\space [pseudonym of J. K. Rowling]\}"
\end{verbatim}
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 edmax:{number}, edmin:{number}, edetal, edtrim:{number} and edprint:{value} (with \ED macro). Example:

editor = "Jan Tomek and Petr Karas",
option = "edprint:{\ED, editors.} edtrim:1",
Output: J. TOMEK and P. KARAS, editors.

If edprint option is not set then \{\ED, \_eds.} or \{\ED, \_ed.} is used depending on the entry language and on the singular or plural of the editor(s).

The ednote field
The 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.

ednote = "Illustrations by Robert \upper{Agarwal}, edited by Tom \upper{Nowak}"
output: Illustrations by Robert AGARWAL, edited by Tom NOWAK.

The \upper command has to be used for Lastnames in the ednote field.

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 titlepost:{value}.

Example:

title = "The Simple Title of The Work",
or
title = "Main Title: Subtitle",
or
title = "Main Title: Subtitle",
option = "titlepost:{Secondary title}"

The output of the last example: Main Title: Subtitle. Secondary title.

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:

edition = "Second",
edition = "2nd",
edition = "2$^{\rm nd}$",
edition = "2."
Output of the last example: 2. ed.

```
edition = "2."
l = "cs",
```

Output: 2. vyd.

Note, that the example `edition = "Second"` may cause problems. If you are using language "cs" then the output is bad: Second vyd. But you can use `editionprint:{⟨value⟩}` option. The the ⟨value⟩ is printed instead of edition field and shortcut. The edition field must be set. Example:

```
edition = "whatever",
option = "editionprint:{Second full revised edition}"
```


You can use \EDN macro in editionprint value. This macro is expanded to the edition value. Example:

```
edition = "Second",
option = "editionprint:{\EDN\space full revised edition}",
or
edition = "Second full revised edition",
option = "editionprint:{\EDN}";
```

The address, publisher, year fields

This is an anachronism from ancient Bib\TeX{} (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:

```
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",
```


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(!), ed-
  note, 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: @BACHELORSTHESIS, @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 @ONLINE entry
  It is intended for online publications.
  Fields: author, title(!), howpublished, ednote, publisher, accessed, doi, url(!), note.

- 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 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].

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 sortedby (or key field
with the same effect). It can be used for sorting instead of the “Lastname Firstname(s)” of the first
author. If the sortedby field is unset then the “Lastname Firstname(s)” is used for sorting normally. Example:

   author = "Světla Čmejrková",
sortedby = "Čmejrkova 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 sortedby = "@", 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 bibTEX 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:

   \_sdef{mt:bib.blue-ray:en} {Blue-ray disc}
   \_sdef{mt:bib.blue-ray:cs} {Blue-ray disk}

Summary of non-standard fields
This style uses the following fields unknown by bibTEX:

   option ... options separated by spaces
   lang ... the language two-letter code of one entry
   ednote ... edition info (secondary authors etc.) or
global data in @MISC-like entries
   citedate ... the date of the citation in year/month/day format
   numbering ... format for volume, number, pages
   isbn ... ISBN
   issn ... ISSN
   doi ... DOI
   url ... URL

Summary of options

   aumax:{⟨number⟩} ... maximum number of printed authors
   aumin:{⟨number⟩} ... number of printed authors if aumax exceeds
   autrim:{⟨number⟩} ... full Firstnames iff number of authors are less than this
   auprint:{⟨value⟩} ... text instead authors list (\AU macro may be used)
   edmax, edmin, edtrim ... similar as above for editors list
   edprint:{⟨value⟩} ... text instead editors list (\ED macro may be used)
   titlepost:{⟨value⟩} ... text after title
   yearprint:{⟨value⟩} ... text instead real year (\YEAR macro may be used)
   editionprint:{⟨value⟩} ... text instead of real edition (\EDN macro may be used)
   urlalso ... the ``available also from'' is used instead ``available from''
   unpublished ... the publisher etc. fields are not mandatory
   nowarn ... no mandatory fields

Other options in the option field are silently ignored.
2.32.7 Implementation of the bib-iso690 style

\_maybedot (alias \: in the style file group) does not put the second dot.

```latex
\_def\_maybedot{\_ifnum\_spacefactor=\_sfcode`\:.\_relax\_else:\_\fi}
\_tmpnum=\_sfcode`\:. \_advance\_tmpnum by-2 \_sfcode`\?:=\_tmpnum
\_sfcode`\?:=\_tmpnum \_sfcode`\!=\_tmpnum
\_let=\_maybedot % prevents from double periods
\_ifx.\_undefined \_let.\=\_maybedot \_fi % for backward compatibility
```

Option field.

```latex
\_CreateField {option}
\_def\_isbiboption#1#2{\_edef\_tmp{\_noexpand\_isbiboptionA{#1}}\_tmp}
\_def\_isbiboptionA#1{\_def\_tmp##1 #1 ##2\_relax{\_if^##2^\_csname iffalse\_ea\_endcsname \_else\_csname iftrue\_ea\_endcsname \_fi}\_ea\_tmp\_biboptionsi #1 \_relax}
\_def\_bibopt[#1]{#2#3}{\_isbiboption{#1}\_iftrue\_def\_tmp{#2}\_else\_def\_tmp{#3}\_fi\_tmp}
\_def\_biboptionvalue#1#2{\_def\_tmp##1 #1:##2 ##3\_relax{\_def#2{##2}}\_ea\_tmp\_biboptionsi #1: \_relax}
\_def\_readbiboptions{\_RetrieveFieldIn{option}\_biboptionsi
\_toks1=\_ea\_biboptionsi\_relax
\_edef\_biboptionsi{\_space \_the\_toks1 \_space \_the\_biboptions \_space}}
```

Formating of Author/Editor lists.

```latex
\_def\_firstauthorformat{\_upper\_Lastname\_bprintc\_Firstname{, *}\_bprintc\_Von{ *}\_bprintc\_Junior{, *}}
\_def\_otherauthorformat{\_bprintc\_Firstname{* }\_bprintc\_Von{* }\_upper\_Lastname\_bprintc\_Junior{, *}}
\_def\_commonname{\_ifnum\_NameCount=1\_firstauthorformat\_else\_ifnum0\_namecount=\_NameCount
\_ifafter\_maybeetal=\_empty \_bibconjunctionand\_else\_fi
\_else , \_fi
\_otherauthorformat
\_fi}
\_def\_authorname{\_ifx\authlist\_undefined \_edef\_authlist{\_Lastname,\_Firstname,\_Von,\_Junior}\_else \_edef\_authlist{\_authlist;\_Lastname,\_Firstname,\_Von,\_Junior}\_fi}
\_def\_editorname{\_ifnum\_NameCount=0\_namecount=\_NameCount
\_ifx\_maybeetal\_empty \_bibconjunctionand\_else\_fi
\_else , \_fi
\_otherauthorformat
\_fi}
```

\_def\_prepareauedoptions#1{\_def\_maybeetal{}\_csname lb@abbreviatefalse\_endcsname
\_biboptionvalue{#1\_max}\_authormax
\_biboptionvalue{#1\_min}\_authormin
\_biboptionvalue{#1\_pre}\_authorpre
\_biboptionvalue{#1\_print}\_authorprint
\_biboptionvalue{#1\_etal}\_Mtext{bib.etal}}\_fi
Preparing \bmark (used when \nonumcitations). The \setbibmark is run at the end of each record. The \authlist includes Lastname,Firstname,Von,Junior of all authors separated by semi-colon (no semi-colon at the end of the list). If \bmark isn’t declared explicitly then we create it by the \createbibmark \langle year \rangle; \langle authors-list \rangle; \ldots; \langle year \rangle. It outputs first Lastname (and adds “et al.” if the second author in the \langle authors-list \rangle is non-empty). Then comma and \langle year \rangle is appended. A user can redefine the \createbibmark macro in the \bibtexhook tokens list, if another \bmark format is needed. The macro \createbibmark must be expandable. See also \texttt{OpTeX} trick 0104.

\def \setbibmark {%
\ifx \authlist \undefined \def \authlist {, ;} \fi
\RetrieveFieldIn {bibmark} \tmp
\ifx \tmp \empty
\RetrieveFieldIn {year} \tmp
\edef \tmp \expanded {\csname \lowercase {\tmp} ; \authlist \endcsname \fi}
\bibmark \csname \lowercase {\tmp} \endcsname
\fi
\}
\def \createbibmark \#1; \#2, \#3; \#4, \#5 \fin {% \#1=year \#2=LastName \#3=FirstName \#4=nextAuthor
\#2 \ifx ^ \#4 ^ \else \Mtext {bib.etal} \fi \ifx ^ \#1 ^ \else , \#1 \fi
\}

\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 { \ifnum \namecount > 1 \Mtext {bib.editors} \else \Mtext {bib.editor} \fi }
\Mtext {\langle identifier \rangle} expands to a phrase by outer document language (no entry language).
\def \setlang #1 {\ifx \#1 \empty \else \setbox 0 = \vbox { \langinput {\#1} } \fi}

\CreateField {lang}
\def \setlang #1 {\ifx \#1 \empty \else \setbox 0 = \vbox { \langinput {\#1} } \fi}
\ifcsname \lang @ \endcsname
\language = \csname \#1 \endcsname \relax
\else \opwarning {No phrases for "\#1" used by \EntryKey in \bib} \fi
\fi
}

\def \fieldalias {key} {sortedby}

\fieldalias {key} {sortedby}
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. The sorting can be used for various other applications, see an example in Op\TeX{} trick 0068.

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{\_sortingdatalatin} implements these rules for the languages with latin alphabets. The groups between commas are not distinguished in the first pass. The second pass distinguishes all characters mentioned in the \texttt{\_sortingdatalatin} (commas are ignored). The order of letters in the \texttt{\_sortingdatalatin} macro is significant for the sorting algorithm.
Characters to be ignored during sorting are declared in `\_ignoredcharsgeneric`. These characters are ignored in the first pass without additional condition. All characters are taken into account in the second pass: ASCII characters with code < 65 are sorted first if they are not mentioned in the \_sortingdata... macro. Others not mentioned characters have undefined behavior during sorting.

```latex
\_def \_ignoredcharsgeneric {.,;?!:'"|()\[<>=+-}
```

Sorting is always processed by rules of a given language. The macros `\_sortingdata(lang-tag)`, `\_ignoredchars(lang-tag)` and `\_compoundchars(lang-tag)` declare these rules. The ⟨lang-tag⟩ is ISO code of the language: en, cs, de, pl, es for example. The English language is implemented here. Other languages are implemented in the `lang-data.opm` file (see section 2.37.4).

The `\_compoundchars(lang-tag)` can declare changes performed before sorting. For example Czech language declares:

```latex
\_let \_sortingdatacs = \_sortingdatalatin \% Czech alphabet is subset of Latin
\_def \_compoundcharscs {ch:^^T Ch:^^U CH:^^V}
```

It transforms two-letters ch to single character ^^T because ch is treated as single compound character by Czech rules and CH is sorted between H and I. See \_sortingdatalatin where ^^T is used. This declaration makes more transformations of Ch and CH too. The declarations of the form x:y in the \_compoundchars(lang-tag) are separated by space.

You can declare a transformation from single letter to more letters too. For example German rules sets ß equal to ss during sorting:

```latex
\_def \_compoundcharsde {ß:ss}
```

If there are two words equal after first pass of sorting: Masse (mass) and Maße (measures) for example, then second pass must decide about the order. DIN 5007, section 6.1 says: ss must be before ß in this case. So, we want to switch off the \_compoundchars declaration for the second pass and use the order of s and ß given in \_sortingdata. This is possible if the \_xcompoundchars(lang-tag) is defined. It has precedence in the second pass of sorting. We declare for German:

```latex
\_def \_xcompoundcharsde {}
```

German rules mention alternative sorting for phone-books or similar lists of names. The letters ä ö ü should be interpreted as ae, oe and ue. So we get Mueller < Müller < Muff. If this rule is not taken into account, we get Mueller < Muff < Müller. The rule can be implemented by:

```latex
```
Because \textasciitilde u < \textasciitilde ü in _sortingdata and because _xcompoundcharsde is empty, we have \textasciitilde Mueller < \textasciitilde Müller after second pass of the sorting.

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 \textasciitilde^I, \textasciitilde^J or \textasciitilde^M because these characters have very specific category codes.

If you created _sortingdata etc. for your language, please, send them to me. I am ready to add them to the file lang-data.omp in a new OpTeX release. See also section 2.37.4.

French sorting rule says: if the words are the same except for accents then accented letters are sorted after unaccented letters but read the words from their end in the second pass. For example correct sorting is: cote < côte < coté < côté. This rule can be activated if the control sequence _secondpass⟨lang-tag⟩ is set to _reversewords. For example, lang-data.omp declares _let_secondpassfr=_reversewords.

Preparing to primary pass is performed by the _setprimarysorting macro implemented here. The (lang-tag) is saved to the _sortinglang macro when sorting is initialized in _dosorting (it is typically derived from current _language value). The _setprimarysorting is called from _dosorting macro and all processing of sorting is in a group. It sets actual _sortingdata, _compoundchars and _ignoredchars if given language declares them. If not then warning will be printed using _nold macro and English data are used. The _lccode of all characters from _sortingdata and _ignoredchars are set. The sorted words will be converted using _compoundchars followed by _lowercase before first pass is run.

Preparing to secondary pass is implemented by the _setsecondarysorting macro.

Strings to be sorted are prepared in \(\langle \text{string} \rangle\) control sequences (to save \TeX memory). The _preparesorting \(\langle \text{string} \rangle\) converts \(\langle \text{string} \rangle\) to _tmpb with respect to the data initialized in _setprimarysorting or _setsecondarysorting. The part of the string after \textasciitilde^\textasciitilde is ignored (you can have the same sorting key for different things) and the compound characters are converted by the _docompound macro.
Macro \_isAleB \langle string1 \rangle \langle string2 \rangle \rangle returns the result of comparison of given two strings to \_ifAleB control sequence. Usage: \_isAleB \langle string1 \rangle \langle string2 \rangle \_ifAleB ... \_else ... \_fi The converted strings (in respect of the data prepared for first pass) must be saved as values of \langle string1 \rangle and \langle string2 \rangle macros. The reason is speed: we don’t want to convert them repeatedly in each comparison. The macro \_testAleB \langle converted-string1 \rangle &\_relax \langle converted-string2 \rangle \&\_relax \langle string1 \rangle \langle string2 \rangle \_ifAleB ... \_else ... \_fi The converted strings (in respect of the data prepared for first pass) must be saved as values of \langle string1 \rangle and \langle string2 \rangle macros. The reason is speed: we don’t want to convert them repeatedly in each comparison. The macro \_testAleB \langle converted-string1 \rangle &\_relax \langle converted-string2 \rangle \&\_relax \langle string1 \rangle \langle string2 \rangle \_ifAleB ... \_else ... \_fi

The \_testAleBsecondary \langle string1 \rangle \langle string2 \rangle is run if the words are equal in the primary pass. It runs \_setsecondarysorting if it was not initialized already. Then prepares compared words to \_tmpa and \_tmpb and corrects them by \_prepsecondpass if needed. Finally, the test is recursively done by the macro \_testAleBsecondaryX \langle converted-string1 \rangle 0 \_relax \langle converted-string2 \rangle 1 \_relax

Merge sort is very effectively implemented by \TeX{} macros. The following code is created by my son Miroslav. The \_mergesort macro expects that all items in \_iilist are separated by a comma when it starts. It ends with sorted items in \_iilist without commas. So \_dosorting macro must prepare commas between items.
The \dosorting list macro redefines \list as sorted \list. The \list have to include control sequences in the form $\backslash\langle c\rangle \langle string\rangle$. These control sequences will be sorted with respect to $\langle string\rangle$ without change of meanings of these control sequences. Their meanings are irrelevant when sorting. The first character $\langle c\rangle$ in $\backslash\langle c\rangle \langle string\rangle$ should be whatever. It does not influence the sorting. Op\TeX uses comma at this place for sorting indexes:

$\langle word1 \rangle, \langle word2 \rangle, \langle word3 \rangle$ ...

The current language (chosen for hyphenation patterns) is used for sorting data. If the macro \sortinglang is defined as $\langle lang-tag\rangle$ (for example \def\sortinglang{de} for German) then this has precedence and current language is not used. Moreover, if you specify \asciisorttrue then ASCII sorting will be processed and all language sorting data will be ignored.

French rules needs reverse reading the words in the second pass. The \reversewords is activated in this case and it adds new job to the macro \prepsecondpass: it reverses the letters in the compared words (saved in \tmpa and \tmpb) by the expandable \sortrevers macro. The \prepsecondpass macro is used in the \testAleBsecondary and it is empty by default.

The \makeindex prints the index. First, it sorts the \iilist second, it prints the sorted \iilist, each item is printed using \printindexitem. We set \leftskip=\iindent and we suppose that each index entry starts by \noindent\hskip-\iindent (see the macro \printii). Then the next lines of the same index entry (if the page list is broken to more pages) is indented by \leftskip=\iindent.
The \printindexitem \verb\langle word \rangle prints one item to the index. If \verb\langle word \rangle is defined then this is used instead real \verb\langle word \rangle (this exception is declared by \verb\iis macro). Else \verb\langle word \rangle is printed by \verb\printii.

Finally, \printii pages prints the value of \verb\langle word \rangle, i.e. the list of pages.

\verb\printii \langle word \rangle& does more intelligent work because we are working with words in the form \verb\langle main-word \rangle/\langle sub-word \rangle/\langle sub-sub-word \rangle. The \verb\everyii tokens register is applied before \verb\noindent. User can declare something special here.

The \newiiletter{\langle letter \rangle}{\langle word \rangle} macro is empty by default. It is invoked if first letter of index entry is changed. You can declare a design between index entries here. You can try, for example:

\begin{verbatim}
def\newiiletter#1#2{
    \bigskip\hbox{\setfontsize{at15pt}\bf #1}\nobreak\medskip}
\end{verbatim}

\verb\definefirstii \langle word \rangle& macro defines \verb\firstii which is used as the \verb\langle letter \rangle parameter of the macro \newiiletter and for testing if the "first letter" of the index entry was changed. The \verb\uppercase of the real first letter is used by default here. You can re-implement \definefirstii if you want. For example, you want to ignore accents above letters for index sub-headers:

\begin{verbatim}
def\definefirstii #1#2&{
    \uppercase\{\def\firstii{#1}\}}
\end{verbatim}

\verb\definefirstii \langle word \rangle& macro defines \verb\firstii which is used as the \verb\langle letter \rangle parameter of the macro \newiiletter and for testing if the "first letter" of the index entry was changed. The \verb\uppercase of the real first letter is used by default here. You can re-implement \definefirstii if you want. For example, you want to ignore accents above letters for index sub-headers:
The implementation of macros \_pgi: because we print \langle \langle of the \_Xindex{The file}. All other variants of indexing macros expand internally to \iindex{The \_iilist \{ word \} is appended to this list. Moreover, we need a mapping from \pg\_iilist \{ word \} to the index. It writes \iindex{word}\{iitype}\} to the .ref file. All other variants of indexing macros expand internally to \index{word}.\}

\_printiipages \{pilist\}\& gets \{pilist\} in the form \{pg\} \{type\}, \{pg\} \{type\}, \ldots \{pg\} \{type\} and it converts them to \{pg\}, \{pg\}, \{from\}--\{to\}, \{pg\} etc. The same pages must be printed only once and continuous consequences of pages must be compressed to the form \{from\}--\{to\}. Moreover, the consequence is continuous only if all pages have the same \{type\}. Empty \{type\} is most common, pages with b \{type\} must be printed as bold and with i \{type\} as italics. Moreover, the \{pg\} mentioned here are \{gpageno\}, but we have to print \{pageno\}. The following macros solve these tasks.

You can re-define \_pgprint \{gpageno\}:\{\{iitype\}\} if you need to implement more \{iitypes\}.\}

The \iindex{word}\{\{iitype\}\} puts one \{word\} to the index. It writes \Xindex{word}\{\{iitype\}\} to the .ref file. All other variants of indexing macros expand internally to \index{word}.\}

The \_Xindex{word}\{\{iitype\}\} stores \{word\} to the \_iilist if there is the first occurrence of the \{word\}. Moreover, we need a mapping from \{gpageno\} to \{pageno\}, because we print \{pageno\} in the index, but hyperlinks are implemented by \{gpageno\}. So, the macro \_pgi: \{gpageno\} is defined as \{pageno\}.\}

The implementation of macros \_i, \_iid, \_iis follows. Note that \_i works in the horizontal mode in order to the \write what's is not broken from the following word. If you need to keep vertical mode,
use \index{\langle word \rangle} directly.
The \texttt{iitype \{\langle type \rangle\}} saves the \langle type \rangle to the \_iitypesaved macro. It is used in the \index macro.

\begin{verbatim}
\def\ii #1 {\leavevmode\def\_tmp(#1)\_ii A #1,,\_def\_iitypesaved{}}
\def\iia #1,\_def\_tmp(#1)\_ea \_iiB \_tmp,,\_else\_iindex{#1}\_fi
\def\iiA #1,{\_if$#1$\_else\def\tmpa{#1}\_ifx\tmpa\_iiatsign \_ea \_iiB \_tmp,,\_else\_iindex{#1}\_fi}
\def\iiatsign{@}
\def\iiB #1,{\_if$#1$\_else \_iiC#1/\relax \_ea \_iiB \_fi}
\def\iiC #1/#2\relax{\_if$#2$\_else\_iindex{#2#1}\_fi}
\def\iid #1 {\_leavevmode\_iindex{#1}\_def\_iitypesaved{}#1\_futurelet\_tmp\_iiD}
\def\iiD{\_ifx\_tmp,,\_else\_ifx\_tmp.\_else\space\_fi}\_fi}
\def\iis #1 #2{{\_def~{ }\_global\_sdef{_,#1}{#2}}\_ignorespaces}
\def\iitypesaved{}
\def\iitype #1{\_def\_iitypesaved{#1}\_ignorespaces}
\public \ii \iid \iis \iitype ;
\end{verbatim}

\section{Footnotes and marginal notes}
\begin{verbatim}
\_newcount \gfnotenum \gfnotenum=0
\_newcount \lfnotenum
\newifi \ifpgfnote
\_def \_fnotenumglobal {\_def \_fnotenum{\_the \gfnotenum}\_pgfnotefalse}
\_def \_fnotenumchapters {\_def \_fnotenum{\_the \lfnotenum}\_pgfnotefalse}
\_def \_fnotenumpages {\_def \_fnotenum{\_trycs{\_fn:\_the \gfnotenum}{?}}\_pgfnotetrue}
\_fnotenumchapters % default are footnotes counted from one in each chapter
\_def \fnotenum{\_fnotenum}
\_let \runningfnotes = \_fnotenumglobal % for backward compatibility
\end{verbatim}

The \texttt{\printfnotemark} prints the footnote mark. You can re-define this macro if you want another design of footnotes. For example

\begin{verbatim}
\_fnotenumpages
\def \_printfnotemark {\_ifcase 0\fnotenum\or
*\or**\or***\or$^\mathbox{†}$\or$^\mathbox{‡}$\or$^\mathbox{††}$\fi}
\_printfnotemarkA\def \_printfnotemark {\_ifcase 0\fnotenum\or
*\or**\or***\or$^\mathbox{†}$\or$^\mathbox{‡}$\or$^\mathbox{††}$\fi}
\_printfnotemarkB\def \_printfnotemark {\_ifcase 0\fnotenum\or
*\or**\or***\or$^\mathbox{†}$\or$^\mathbox{‡}$\or$^\mathbox{††}$\fi}
\end{verbatim}

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 \texttt{\printfnotemarkA} and \texttt{\printfnotemarkB}.
The \texttt{\fnotelinks\langle colorA\rangle\langle colorB\rangle} 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(pgfnote) in the macro \fn: (fnenum). Each \_Xpage macro sets the \_lfnotenum to zero.

The \fnotetext macro is simple, \fnotemark and \fnotetext does the real work.

By default \mnote{(text)} are in right margin at odd pages and they are in left margin at even pages. The \mnote macro saves 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 #2 trick returns true if \(\textsc{pageno}\) has a numeric type and false if it is a non-numeric type (Roman numeral, for example). We prefer to use \(\textsc{pageno}\), but only if it has the numeric type. We use \(\textsc{gpageno}\) 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 \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{dimen}{text} is possible too, but public \mnoteskip is kept for backward compatibility.

The \mnoteA macro does the real work. The \lrmnote{left}{right} uses only first or only second parameter depending on the left or right marginal note.

\begin{verbatim}
161 \long\def\mnoteA #1{\increment\mnotenum
162 \ifx\mnotesfixed\undefined
163 \edef\mnotesfixed{\csname _mn:\the\mnotenum\endcsname}%
165 \else
166 \opwarning{unknown \noexpand\mnote side. TeX me again}\openref
167 \increment\unresolvedrefs
168 \def\mnotesfixed{\right}%
169 \fi\fi
170 \hbox to0pt{\wref\Xmnote{}\everypar={}%
171 \lrmnote{\kern-\mnotesize \kern-\mnoteindent}{\kern\hsize \kern\mnoteindent}%
172 \vbox to0pt{\vss \setbox0=\vtop{\hsize=\mnotesize
173 \lrmnote{\leftskip=0pt plus 1fill \rightskip=0pt}%
174 \{\the\everymnote\noindent\endgraf}%)%
175 \dp0=0pt \box0 \kern\mnoteskip \global\mnoteskip=0pt\hss}%
176 \def\lrmnote#1#2{\ea\ifx\mnotesfixed\left #1\else #2\fi}
\end{verbatim}

We don't want to process \fnote, \fnotemark, \mnote in TOC, headlines nor outlines.

\begin{verbatim}
185 \regmacro {\def\fnote#1{}} {\def\fnotemark#1{}} {\def\mnote#1{}}
186 \regmacro {\def\fnotemark#1{}} {\def\fnotemark#1{}} {\def\mnote#1{}}
187 \regmacro {\def\fnote#1{}} {\def\mnote#1{}} {\def\mnote#1{}}
\end{verbatim}

\section{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 op-slides.pdf which is an example of the presentation.

\subsection{2.35.1 \report and \letter styles}

\begin{verbatim}
3 \codedecl \report (Basic styles of OpTeX <2021-03-10>) % preloaded in format
\end{verbatim}

We define auxiliary macro first (used by the \address macro)
The \boxlines{line-1}{col}{line-2}{col}...{line-n}{col} returns to the outer vertical mode a box with \line, next box with \line, 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 \col is set active and \everypar starts \ hbox{ and active \col} closes this \ hbox by \.

\begin{verbatim}
16 \def\boxlines{
17 \def\boxlinesE{\ifhmode\egroup\empty\fi}%
18 \def\nl\boxlinesE%
19 \egroup \lccode`\-=\``\lowercase{\egroup}_{let-}\boxlinesE%
20 \everypar{\setbox0=\lastbox\endgraf}
\end{verbatim}
The \report style initialization macro is defined here.

\begin{verbatim}
def_report{
  \typosize[11/13.2]
  \vsize=\dimexpr \topskip + 52\baselineskip \relax % added 2020-03-28
  \let\titfont=\chapfont
  \titskip=3ex
  \edef\author{\removelastskip \bigskip
    \leftskip=0pt plus1fill \rightskip=\leftskip \it \noindent \par}
  \nobreak \bigskip
}
\public \author ;
\parindent=1.2em \iindent=\parindent \ttindent=\parindent
\footline={\global\footline={\hss\rmfixed\folio\hss}}
\end{verbatim}

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.

\begin{verbatim}
def_letter{
  \def\address{\vtop\bgroup\boxlines \parskip=Opt \let\par=\egroup}
  \def\subject{\bf \mtext{subj}: }
  \address \subject ;
  \typosize[11/14]
  \vsize=\dimexpr \topskip + 49\baselineskip \relax % added 2020-03-28
  \parindent=0pt
  \parskip=\medskipamount
  \nopagenumbers
}
\public \letter \report ;
\end{verbatim}

The \slides macro reads macro file \texttt{slides.opm}, see the section \ref{sec:slides}.

\begin{verbatim}
def\slides{\par
  \opinput{slides.opm}
  \adef*(\relax\ifmmode*\else\startitem\fi}
}
\public \slides ;
\end{verbatim}

\subsection{\texttt{\slide}s style for presentations}

Default margins and design is declared here. The \texttt{\ttfnt} is scaled by mag1.15 in order to balance the ex height of Helvetica (Heros) and LM fonts Typewriter. The \texttt{\begtt...\endtt} verbatim is printed by smaller text.

\begin{verbatim}
def\wideformat{\margins/1 (14,14,10,3)mm % landscape A5 format
  \def\fontnamegen{\undefined \fontname[Heros]}
  \def\ttfnt=\undefined \fontvardef\ttfnt{\setfontsize{mag1.15}\tt}
  \fi
  \tpsize[16/19]
  \urlfont{}
  \everytt=[\tpsize[13/16] \advance\hsize by10mm}
  \fontdef\fixbf{\bf}
  \nopagenumbers
  \parindent=Opt
  \ttindent=5mm
\end{verbatim}
The bottom margin is set to 3mm. If we use 1mm, then the baseline of the \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 14mm for narrow pages or 16mm for wide pages.

The \subtit is defined analogically like \tit.

The \pshow⟨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 \_scolor.
When \texttt{\slideshow} is active then each page is opened by \texttt{\setbox\_slidepage\=vbox\bgroup} (roughly speaking) and closed by \texttt{\egroup}. The material is \texttt{\unvboxed} and saved for the usage in the next usage if \texttt{\pg+} is in process. The \texttt{\_slidelayer} is incremented instead \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{\langle layered-text \rangle} is saved to \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 the \texttt{\_printlayers}.
\_newcount\_layernum \_newcount\_maxlayers
\maxlayers=0
\long\def\layersactive #1 #2\endlayers{%  
\par\penalty0\_egroup\_egroup
\gdef\_layertext\{\_settinglayer#2}%
\global\maxlayers=#1
\_setbox\_slidepage=\_vbox\_bgroup\_bgroup
\_setbox0=\_vbox\{\layernum=1 \_globaldefs=-1 \_layertext\_endgraf\}
\prevdepth=\dp0
}%
\public \subtit \slideshow \pg \wideformat \use \pshow \layernum ;
\slideopen 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. \_destbox is saved value of the original \_destbox which is redefined to do only \_addto\_destboxes. 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\{\langle\label\rangle\}\wlabel\langle\text\rangle in the last part of the page (before \pg\; ) use \pgref instead of \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 \label macro). Moreover, the warning about illegal use of \bib, \usebib in \layers environment is activated.

Default \layers\{\langle\layertext\rangle\} with \layernum=\langle\num\rangle+1 because we need the result after last layer is processed.
We must to redefine \texttt{\fnotenumpages} 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 \texttt{\pg}; macro, so we can reset the footnote counter by this macro.

\begin{verbatim}
27 \def \fnotenumpages {def\fnotenum{\the\lfnotenum}\pgfnotefalse
28  \def\lfnotenumreset{\global\lfnotenum=0 }}
29 \let \lfnotenumreset=\relax
30 \public \fnotenumpages ;
\end{verbatim}

2.36 Logos

\begin{verbatim}
\_protected\def \_TeX {T\_kern-.1667em\lower.5ex\hbox{E}\_kern-.125emX\_ignoreslash}
\_protected\def \_OpTeX {Op\_kern-.1em\TeX}
\_protected\def \_LuaTeX {Lua\_TeX}
\_protected\def \_XeTeX {X\_kern-.125em\phantom E\_pdfsave\rlap{\_pdfscale{-1}{1}\_lower.5ex\hbox{E}}\_pdfrestore \_kern-.1667em \_TeX}
\_protected\def \_ignoreslash {\isnextchar/\_ignoreit{}}
\_public \TeX \OpTeX \LuaTeX \XeTeX \ignoreslash ;
\end{verbatim}

\begin{verbatim}
3 \_codedecl \TeX {Logos TeX, LuaTeX, etc. <2020-02-28>} % preloaded in format
\end{verbatim}

Despite plain \TeX each macro for logos ends by \texttt{\ignoreslash}. This macro ignores the next slash if it is present. You can use \texttt{\TeX/ like this} for protecting the space following the logo. This is visually more comfortable. The macros \TeX, \OpTeX, \LuaTeX, \XeTeX are defined.

The \texttt{\slantcorr} macro expands to the slant-correction of the current font. It is used to shifting A if the \LaTeX logo is in italic.

\begin{verbatim}
\_protected\def \LaTeX{\_tmpdim=.42ex L\_kern-.36em \_kern \_slantcorr % slant correction
29 \_raise \_tmpdim \_hbox{\_thefontscale[710]{A}}\_kern-.15em \_kern-\_slantcorr \_TeX}
\_def \_slantcorr{\_ea\_ignorept \_the\_fontdimen1\_font \_tmpdim}
\_public \LaTeX ;
\end{verbatim}

\begin{verbatim}
\_protected\def \LaTeX{\_tmpdim=.42ex L\_kern-.36em \_kern \_slantcorr % slant correction
29 \_raise \_tmpdim \_hbox{\_thefontscale[710]{A}}\_kern-.15em \_kern-\_slantcorr \_TeX}
\_def \_slantcorr{\_ea\_ignorept \_the\_fontdimen1\_font \_tmpdim}
\_public \LaTeX ;
\end{verbatim}

\texttt{\OPmac, \CS and \csplain logos.}

The expandable versions of logos used in Outlines need the expandable \texttt{\ignnslash} (instead of the \texttt{\ignoreslash}).

\begin{verbatim}
\_def\_ignnslash#1{\ifx/#1\else #1\fi}
\_regmacro (){} % conversion for PDF outlines
\_def\TeX\_ignnslash{\_def\OpTeX{\OpTeX\_ignnslash}}
\_def\LuaTeX\_ignnslash{\_def\XeTeX{\XeTeX\_ignnslash}}
\_def\LaTeX\_ignnslash{\_def\PuMac{\PuMac\_ignnslash}}
\_def\CS\_ignnslash{\_def\CSplain{\CSplain\_ignnslash}}
\_public \OPmac \CS \csplain ;
\end{verbatim}

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 \texttt{\lccode} and \texttt{\uccode}. The \texttt{\catcode} above the code 127 is not set, i.e. the \texttt{\catcode=12} for all codes above 127.

199
The file `UnicodeData.txt` is read if this file exists in your TeX distribution. The format is specified at [http://www.unicode.org/L2/L1999/UnicodeData.html](http://www.unicode.org/L2/L1999/UnicodeData.html). We read only `Ll` (lowercase letters), `Lu` (uppercase letters) and `Lo` (other letters) and set appropriate codes. The scanner of `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`.

---

### 2.37.2 Multilingual phrases and quotation marks

We can declare such language-dependent words by

\begin{verbatim}
\_sdef{\_mt:\langle phrase-id\rangle:\langle lang-tag\rangle}{\langle \text{label} \rangle}
\end{verbatim}

We use more “compact” macro `\_lang` `\langle lang-tag \rangle \langle chapter \rangle \langle table \rangle \langle figure \rangle \langle subject \rangle` for declaring them.
The used when the first character is allow to quote the quotes mark itself by the quotes parameter is read in verbatim mode and retokenized again by \quoteschars

The should be declared by including its quotation marks used for depending on the actual selected language. For example, \quotes here. Languages in general provide the \enquotes of these two types of quotes. \normalquotes and are created. User can declare quotes by \"macros

Remember, that they are used in another meaning when the document (for example \quotes

This example declares date format for English where \langle

\_today macro needs auto-generated words for each name of the month.

\_monthw \langle{lang-tag} \langle{January} \langle{February} \ldots \langle{December} is used for decaring them.

The language-dependent format for printing date should be declared like

\_sdef{\_mt:today:en}{\_mtext{m\_the\_month} \_the\_day, \_the\_year}

This example declares date format for English where \langle{lang-tag} is en.

Quotes should be tagged by \"(text)" and \' (text)' if \langle{iso-code}quotes is declared at beginning of the document (for example \enquotes). If not, then the control sequences \" and \' are undefined. Remember, 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\langle{clq}\langle{crq}\langle{clq}\langle{crq}, where \langle{clq} \ldots \langle{crq} are normal quotes and \langle{clq} \ldots \langle{crq} are alternative quotes. or use \altquotes to swap between the meaning of these two types of quotes. \enquotes, \csquotes, \frquotes, \dequotes, \skquotes are defined here. Languages in general provide the \quotes declaration macro. It declares the quotation marks depending on the actual selected language. For example, \eslang \quotes declares Spanish language including its quotation marks used for \"(text)" and \'(text)'. The language-dependent quotation marks should be declared by \quotemarks{lang-tag} \{\langle{clq}\langle{crq}\langle{clq}\langle{crq}\}

The \quoteschars\langle{qq}\langle{rq}\langle{lj}\langle{rq} defines \" and \' as \qqA in normal mode and as expable 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 \scantextokens. We want to allow to quote the quotes mark itself by \"\text{"}\. This is the reason why the sub-verbatim mode is used when the first character is \{ in the parameter.

The \qqA\qqB\langle{qq}\langle{rq}\langle{lj}\langle{rq} and \ as \qqA\qqC\langle{lj}\langle{rq}. The \qqA\qqB\langle{clq}\langle{crq}\langle{clq}\langle{crq}\} runs \qqB\langle{qq}\langle{rq}\langle{text} ".
The \regquotes\"(\langle L\rangle \langle R\rangle)\ does \def\#1\{}{\langle L\rangle#1\langle R\rangle}\ for outlines but the " separator is active (because " and ' are active in \pdfunidef).

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 \activequotes\ macro and leave quotes without the first backslash. First, declare \{iso-code\}quotes, then \altquotes\ (if needed) and finally \activequotes\.

\langlist\{Languages declaration <2022-10-11>\} % preloaded in format

\_preplang\ (lang-id) \LongName\ (lang-tag) \hyph-tag\ (lr-hyph) declares a new language. The parameters (separated by space) are

- \{lang-id\}: language identifier. It should be derived from ISO 639-1 code but additional letters can be eventually added because \{lang-id\} must be used uniquely in the whole declaration list. The \_preplang\ macro creates the language switch \{lang-id\}\lang\ and defines also \{lang-id\}\lang\ as a macro which expands to \{lang-id\}\lang. For example, \_preplang\ cs Czech ... creates \cslang\ as the language switch and defines \def\cslang\{\cslang\}.
- \{LongName\}: full name of the language.
- \{lang-tag\}: language tag, which is used for setting language-dependent phrases and sorting data. If a language have two or more hyphenation patterns but a single phrases set, then we declare this language more than once with the same \{lang-tag\} but different \{lang-hyph\}.
- \{hyph-tag\}: a part of the file name where the hyphenation patterns are prepared in Unicode. The full file name is \texttt{\textbackslash hyph}\{-\texttt{\textbackslash hyph-tag}\}.tex. If \{hyph-tag\} is \{\} then no hyphenation patterns are loaded.
- \{lr-hyph\}: two digits, they denote \lefthyphenmin\ and \righthyphenmin\ values.

\_preplang\ allocates a new internal number by \_newlanguage\{-\{lang-id\}\Patt\} which will be bound to the hyphenation patterns. But the patterns nor other language data are not read at this moment.

The \{lang-id\}\lang\ is defined as \langinit\lang. When the \{lang-id\}\lang\ switch is used firstly in a document then the language is initialized, i.e. hyphenation patterns and language-dependent data are read. The \{lang-id\}\lang\ is re-defined itself after such initialization. \_preplang\ does also \def\_ulan:\{longname\} \{\{lang-id\}\}, this is needed for the \uselanguage\ macro.

The \_preplang\ macro adds \{lang-id\}\{-\{LongName\}\} to the \langlist\ macro which is accessible by \langlist. It can be used for reporting declared languages.
All languages with hyphenation patterns provided by \texttt{TeXlive} are declared here. The language switches \texttt{\_preplang} and \texttt{\_lang} and many others are declared. You can declare more languages by \texttt{\_preplang} in your document, if you want. The usage of \texttt{\_preplang} with \texttt{⟨lang-id⟩} already declared is allowed. The language is re-declared in this case. This can be used in your document before first usage of the \texttt{⟨lang-id⟩} switch.

<table>
<thead>
<tr>
<th>lang-id</th>
<th>LongName</th>
<th>lang-tag</th>
<th>hyph-tag</th>
<th>lr-hyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>enus</td>
<td>USenglishmax</td>
<td>en</td>
<td>en-us</td>
<td>23</td>
</tr>
<tr>
<td>engb</td>
<td>UKenglish</td>
<td>en</td>
<td>en-gb</td>
<td>22</td>
</tr>
<tr>
<td>be</td>
<td>Belarusian</td>
<td>be</td>
<td>be</td>
<td>22</td>
</tr>
<tr>
<td>bg</td>
<td>Bulgarian</td>
<td>bg</td>
<td>bg</td>
<td>22</td>
</tr>
<tr>
<td>ca</td>
<td>Catalan</td>
<td>ca</td>
<td>ca</td>
<td>22</td>
</tr>
<tr>
<td>hr</td>
<td>Croatian</td>
<td>hr</td>
<td>hr</td>
<td>22</td>
</tr>
<tr>
<td>cs</td>
<td>Czech</td>
<td>cs cs</td>
<td>cs cs</td>
<td>23</td>
</tr>
<tr>
<td>da</td>
<td>Danish</td>
<td>da da</td>
<td>da da</td>
<td>22</td>
</tr>
<tr>
<td>nl</td>
<td>Dutch</td>
<td>nl nl</td>
<td>nl nl</td>
<td>22</td>
</tr>
<tr>
<td>et</td>
<td>Estonian</td>
<td>et et</td>
<td>et et</td>
<td>23</td>
</tr>
<tr>
<td>fi</td>
<td>Finnish</td>
<td>fi fi</td>
<td>fi fi</td>
<td>22</td>
</tr>
<tr>
<td>fim</td>
<td>schoolFinnish</td>
<td>fi fi-x-school</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>fr</td>
<td>French</td>
<td>fr fr</td>
<td>fr fr</td>
<td>22</td>
</tr>
<tr>
<td>de</td>
<td>nGerman</td>
<td>de de-1996</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>deo</td>
<td>oldGerman</td>
<td>de de-1901</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>gsw</td>
<td>swissGerman</td>
<td>de de-ch-1901</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>elm</td>
<td>monoGreek</td>
<td>el el-monoton</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>elp</td>
<td>Greek</td>
<td>el el-polyton</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>fis</td>
<td>schoolFinnish</td>
<td>fi fi-x-school</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>ga</td>
<td>Irish</td>
<td>ga ga</td>
<td>ga ga</td>
<td>23</td>
</tr>
<tr>
<td>it</td>
<td>Italian</td>
<td>it it</td>
<td>it it</td>
<td>22</td>
</tr>
<tr>
<td>la</td>
<td>Latin</td>
<td>la la</td>
<td>la la</td>
<td>22</td>
</tr>
<tr>
<td>lac</td>
<td>classicLatin</td>
<td>la la-x-classic</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>lal</td>
<td>liturgicalLatin</td>
<td>la la-x-liturgic</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>lv</td>
<td>Latvian</td>
<td>lv lv</td>
<td>lv lv</td>
<td>22</td>
</tr>
<tr>
<td>lt</td>
<td>Lithuanian</td>
<td>lt lt</td>
<td>lt lt</td>
<td>22</td>
</tr>
<tr>
<td>mk</td>
<td>Macedonian</td>
<td>mk mk</td>
<td>mk mk</td>
<td>22</td>
</tr>
<tr>
<td>pl</td>
<td>Polish</td>
<td>pl pl</td>
<td>pl pl</td>
<td>22</td>
</tr>
<tr>
<td>pt</td>
<td>Portuguese</td>
<td>pt pt</td>
<td>pt pt</td>
<td>22</td>
</tr>
<tr>
<td>ro</td>
<td>Romanian</td>
<td>ro ro</td>
<td>ro ro</td>
<td>22</td>
</tr>
<tr>
<td>rm</td>
<td>Romanian</td>
<td>rm rm</td>
<td>rm rm</td>
<td>22</td>
</tr>
<tr>
<td>ru</td>
<td>Russian</td>
<td>ru ru</td>
<td>ru ru</td>
<td>22</td>
</tr>
<tr>
<td>sr</td>
<td>Serbian</td>
<td>sr-latm</td>
<td>sh-latn</td>
<td>22</td>
</tr>
<tr>
<td>src</td>
<td>SerbianCyril</td>
<td>sr-cyrll</td>
<td>sh-cyrll</td>
<td>22</td>
</tr>
<tr>
<td>sk</td>
<td>Slovak</td>
<td>sk sk</td>
<td>sk sk</td>
<td>23</td>
</tr>
<tr>
<td>sl</td>
<td>Slovenian</td>
<td>sl sl</td>
<td>sl sl</td>
<td>22</td>
</tr>
<tr>
<td>es</td>
<td>Spanish</td>
<td>es es</td>
<td>es es</td>
<td>22</td>
</tr>
<tr>
<td>sv</td>
<td>Swedish</td>
<td>sv sv</td>
<td>sv sv</td>
<td>22</td>
</tr>
<tr>
<td>uk</td>
<td>Ukrainian</td>
<td>uk uk</td>
<td>uk uk</td>
<td>22</td>
</tr>
<tr>
<td>cy</td>
<td>Welsh</td>
<td>cy cy</td>
<td>cy cy</td>
<td>23</td>
</tr>
<tr>
<td>af</td>
<td>Afrikaans</td>
<td>af af</td>
<td>af af</td>
<td>12</td>
</tr>
<tr>
<td>hy</td>
<td>Armenian</td>
<td>hy hy</td>
<td>hy hy</td>
<td>12</td>
</tr>
<tr>
<td>as</td>
<td>Assamese</td>
<td>as as</td>
<td>as as</td>
<td>11</td>
</tr>
<tr>
<td>eu</td>
<td>Basque</td>
<td>eu eu</td>
<td>eu eu</td>
<td>22</td>
</tr>
<tr>
<td>bn</td>
<td>Bengali</td>
<td>bn bn</td>
<td>bn bn</td>
<td>11</td>
</tr>
<tr>
<td>nb</td>
<td>Bokmal</td>
<td>nb nb</td>
<td>nb nb</td>
<td>22</td>
</tr>
<tr>
<td>cop</td>
<td>Coptic</td>
<td>cop cop</td>
<td>cop cop</td>
<td>11</td>
</tr>
<tr>
<td>cu</td>
<td>churchSlavonic</td>
<td>cu cu</td>
<td>cu cu</td>
<td>12</td>
</tr>
<tr>
<td>eo</td>
<td>Esperanto</td>
<td>eo eo</td>
<td>eo eo</td>
<td>22</td>
</tr>
<tr>
<td>ethi</td>
<td>Ethiopic</td>
<td>ethi mul-ethi</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>fur</td>
<td>Friulian</td>
<td>fur fur</td>
<td>fur fur</td>
<td>22</td>
</tr>
<tr>
<td>gl</td>
<td>Galician</td>
<td>gl gl</td>
<td>gl gl</td>
<td>22</td>
</tr>
<tr>
<td>ka</td>
<td>Georgian</td>
<td>ka ka</td>
<td>ka ka</td>
<td>12</td>
</tr>
<tr>
<td>gu</td>
<td>Gujarati</td>
<td>gu gu</td>
<td>gu gu</td>
<td>11</td>
</tr>
<tr>
<td>hi</td>
<td>Hindi</td>
<td>hi hi</td>
<td>hi hi</td>
<td>11</td>
</tr>
<tr>
<td>id</td>
<td>Indonesian</td>
<td>id id</td>
<td>id id</td>
<td>22</td>
</tr>
<tr>
<td>ia</td>
<td>Interlingua</td>
<td>ia ia</td>
<td>ia ia</td>
<td>22</td>
</tr>
</tbody>
</table>
\_preplangmore \langle lang-id\rangle\{space\}\{text\}\} declares more activities of the language switch. The \{text\} is processed whenever \_\langle lang-id\rangle\lang is invoked. If \_preplangmore is not declared for given language then \_langdefault is processed.

You can implement selecting a required script for given language, for example:

\_preplangmore ru { \_nonfrenchspacing \_setff\{script=cyrl\}\selectcyrlfont}
\_addto\langdefault { \_setff\{}\selectlatnfont}

The macros \selectcyrlfont and \selectlatnfont are not defined in OpTEX. If you follow this example, you have to define them after your decision what fonts will be used in your specific situation.

The \_langreset is processed before macros declared by \_preplangmore or before \_langdefault. If you set something for your language by \_preplangmore then use \def\langreset{\{settings\}} in this code too in order to return default values for all other languages. See cs part of lang-data.opm file for an example.

The list of declared languages are reported during format generation.

Each language switch \_\langle lang-id\rangle\lang defined by \_preplang has its initial state
\_langinit \_\langle switch\rangle \_\langle lang-id\rangle\{\{LongName\}\{lang-tag\}\{hyph-tag\}\{lr-hyph\}\}.

The \_langinit macro does:
• The internal language \langle number \rangle is extracted from \_the\langle lang-id\rangle Patt.
• \texttt{\def \_lan:(number) {\langle lang-tag \rangle}} for mapping from \texttt{language} number to the \langle lang-tag \rangle.
• loads \texttt{hyph-\langle hyph-tag \rangle.tex} file with hyphenation patterns when \texttt{language=number}.
• loads the part of \texttt{lang-data.opm} file with language-dependent phrases using \texttt{\_langinput}.
• \texttt{\def \_\langle lang-id\rangle lang {\_uselang{\langle lang-id\rangle}} \langle lr-hyph \rangle}, i.e. the switch redefines itself for doing a “normal job” when the language switch is used repeatedly.
• Runs itself (i.e. \_\langle lang-id\rangle lang) again for doing the “normal job” firstly.

```
\_def \_lan:(number) {
  \langle lang-tag \rangle
}
```

2.37.4 Data for various languages

The “language data” include declarations of rules for sorting (see section 2.33), language-dependent phrases and quotation marks (see section 2.37.2). The language data are collected in the single \texttt{lang-data.opm} file. Appropriate parts of this file is read by \texttt{\_langinput{\langle lang-tag \rangle}}. First few lines of the file looks like:

```
\_codedecl \_langdata (Language dependent data <2022-10-11>) % only en, cs preloaded in format
\_langdata en {English} % -----------------------------------------------
\_langw en Chapter Table Figure Subject
\_langb en {, and } { et al.} {\textendash} {	extemdash} {Vol.-} {No.-} {pp.-} {\textendash} {\textendash} {,\textendash} {,\textendash}
\_monthw en January February March April May June
\_monthv en January February March April May June
\_quotationmarks en {"\textendash"}
\_let \_sortingdataen = \_sortingdatalatin % set already, see section 2.33, makeindex.opm
```

The \texttt{\_uselanguage {\langle LongName \rangle}} macro is defined here (for compatibility with e-plain users). Its parameter is case insensitive.

```
\_def \_uselanguage#1{%
  \_def \_tmp{#1}%
  \_lowercase{\_cs{\_trycs{\_ulan:#1}{0x}lang}}
}
```

```
\_langdata cs (Czech) % -----------------------------------------------
```

205
There are analogical declaration for more languages here. Unfortunately, this file is far for completeness. I welcome you send me a part of declaration for your language.

If your language is missing in this file then a warning is reported during language initialization. You can create your private declaration in your macros (analogical as in the lang-data.opm file but without the \_langdata prefix). Then you will want to remove the warning about missing data. This can be done by \_nolanginput{⟨lang-tag⟩} given before initialization of your language.

The whole file lang-data.opm is not preloaded in the format because I suppose a plenty languages here and I don’t want to waste the \TeX memory by these declarations. Each part of this file prefixed by \_langdata{⟨lang-tag⟩} \{⟨LongName⟩\} is read separately when \_langinput{⟨lang-tag⟩} is used. And it is used in the \_langinit macro (i.e. when the language is initialized), so the appropriate part of this file is read automatically on demand.

If the part of the lang-data.opm concerned by ⟨lang-tag⟩ is read already then \_li:⟨lang-tag⟩ is set to R and we don’t read this part of the file again.

\_def\_langinput #1{\unless \ifcsname _li:#1\_endcsname
\_bgroup\_edef\_tmp{\_noexpand\langdata #1}\_everyeof\_ea{\_tmp{}}\_long\_ea\def\_ea\tmp \_ea##\_ea1\_tmp{\_readlangdata{#1}}\_globaldefs=1\_ea\tmp \_input{lang-data.opm}\_ea\glet \csname _li:#1\_endcsname R\%\_egroup\_fi\fi
\_def\_readlangdata #1\#2\{%\_unless \ifs#2\_opwarning{Missing data for language "#1" in lang-data.opm}\%\_else \vlog\{Reading data for the language #2 (#1)\}\%\_fi\}
2.38 Other macros

Miscellaneous macros are here.

\useOpTeX{} and \useoptex{} are declared as \relax{}.

The \lastpage{} and \totalpages{} get the information from the \currpage{}. The \Xpage{} from .ref file sets the \currpage{}.

We need \uv{}, \clqq{}, \crqq{}, \flqq{}, \frqq{}, \uslang{}, \ehyph{}, \chyph{}, \shyph{}, for backward compatibility with ČSplain. Codes are set according to Unicode because we are using Czech only in Unicode when Lua\TeX{} is used.

\vspace{1cm}

The \letfont{} was used in ČSplain instead of \fontlet{}.

Non-breaking space in Unicode.

Old macro packages need these funny control sequences. We don't use them in new macros.
We allow empty lines in math formulae. It is more comfortable.

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)] or \lipsum[(number)] (i.e. dot after the parameter). The first case prints the paragraph \langle number \rangle without the final \_par and the second case prints only first sentence from the paragraph \langle number \rangle using \_lipsumdot.

Selected macros from OpTeX tricks are registered using \_regtrick\langle cs-name\rangle. The \langle cs-name\rangle is defined as \loadtrick\langle cs-name\rangle \langle cs-name\rangle. When a user runs such a registered \langle cs-name\rangle then \loadtrick\langle cs-name\rangle reads the appropriate code from the file optex-tricks.opm and the \langle cs-name\rangle is redefined. Finally, \langle cs-name\rangle is run again.

The optex-tricks.opm file includes blocks started by \_trick followed by declarations \langle cs-names\rangle followed by \_endinput of the file. The file is read inside temporary \vbox with \_globaldefs=1 because it can be used inside horizontal mode and/or inside a group. The optextrick name space is used during reading the code from the file. Only registered control sequences are re-defined directly in user name space.

You can load a code chunk by \loadtrick \langle cs-name\rangle. This command doesn’t run the \langle cs-name\rangle, only loads the appropriate code. It should be usable if you want to load the code before the first usage of the \langle cs-name\rangle.
LuaTeX version 1.14 and newer provides \partokenname which allows to specify something different than \par at empty lines. We set \par (see bellow) in OpTeX version 1.04+ and newer. Some macros were rewritten due to this change. And we copy old versions of these changed macros here in order to allow to use older LuaTeX versions where \partokenname is not provided.

Note that your macros where a parameter is separated by the empty line must be changed too. Use \def\macro #1\par{...} instead of \def\macro #1\par{...}.

We set \partokenname to \par in order to keep the name \par in the public namespace for end users. I.e. a user can say \def\par{paragraph} for example without crash of processing the document. See section 2.2.1 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.
2.39 Lua code embedded to the format

The file \texttt{optex.lua} is loaded into the format in \texttt{optex.ini} as byte-code and initialized by \texttt{\everyjob}, see section 2.1.

The file implements part of the functionality from \texttt{luatexbase} namespace, nowadays defined by \LaTeX{} kernel. \texttt{luatexbase} deals with modules, allocators, and callback management. Callback management is a nice extension and is actually used in Op\TeX{}. 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 2.39.5.

\begin{verbatim}
local fmt = string.format

2.39.1 General

Define namespace where some Op\TeX{} functions will be added.

\begin{verbatim}
local optex = _ENV.optex or {}
_ENV.optex = optex

Error function used by following functions for critical errors.

\begin{verbatim}
local function err(message)
  error("\error: \"..message..\"\n")
end

For a \texttt{\chardef'd}, \texttt{\countdef'd}, etc., csname return corresponding register number. The responsibility of providing a \texttt{\XXdef'd} name is on the caller.

\begin{verbatim}
local function registernumber(name)
  return token.create(name).index
end
_ENV.registernumber = registernumber
optex.registernumber = registernumber

MD5 hash of given file.

\begin{verbatim}
function optex.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
\end{verbatim}

2.39.2 Allocators

\begin{verbatim}
local alloc = _ENV.alloc or {}
_ENV.alloc = alloc

An attribute allocator in Lua that cooperates with normal Op\TeX{} allocator.

\begin{verbatim}
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_ml("log", "\"..name..\"=\attribute..\"\n")
    attributes[name] = cnt
    return cnt
  end
end
\end{verbatim}
\end{verbatim}
\end{verbatim}

210
Allocator for Lua functions (“pseudoprimitives”). It passes variadic arguments (“…”) like "global" to `token.set_lua`.

```lua
local function_table = lua.get_functions_table()
local function define_lua_command(csname, fn, ...) local luafnalloc = #function_table + 1
token.set_lua(csname, luafnalloc, ...) -- WARNING: needs LuaTeX 1.08 (2019) or newer
function_table[luafnalloc] = fn
end
_ENV.define_lua_command = define_lua_command
optex.define_lua_command = define_lua_command
```

### 2.39.3 Callbacks

Save `callback.register` function for internal use.

```lua
local callback = _ENV.callback or {}
_ENV.callback = callback

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",
stop_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,
list = true,
}

212
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
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
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
end

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
  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 overridden by using `add_to_callback`.

```lua
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 luatex 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 
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).

```lua
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 L\(\TeX\) through luatexbase namespace. Needed for luaotfload.

```lua
_ENV.luatexbase = {
    registernumber = registernumber,
}```
attributes = attributes,
-- `provides_module` is needed by older version of luaotfload
provides_module = function() end,
new_attribute = alloc.new_attribute,
callback_descriptions = callback.callback_descriptions,
create_callback = callback.create_callback,
add_to_callback = callback.add_to_callback,
remove_from_callback = callback.remove_from_callback,
call_callback = callback.call_callback,
callbacktypes = {},

\tracingmacros\ callback registered. Use \tracingmacros=3 or \tracingmacros=4 if you want to see the result.

\text{callback.add_to_callback("input_level_string", function(n)\nif tex.tracingmacros > 3 then
return \"[\ .. n .. \]\"
elseif tex.tracingmacros > 2 then
return \"~\ .. string.rep("\ .",n)\nelse
return \"\nend\nend, ",\tracingmacros\")}

\section{Management of PDF page resources}

Traditionally, pdfTeX allowed managing PDF page resources (graphics states, patterns, shadings, etc.) using a single toks register, \pdfrsrc. This is insufficient due to the expected PDF object structer and also because many “packages” want to add page resources and thus fight for the access to that register. We add a finer alternative, which allows adding different kinds of resources to a global page resources dictionary. Note that some resource types (fonts and XObjects) are already managed by Lua\TeX\ and shouldn’t be added!

XObject forms can also use resources, but there are several ways to make Lua\TeX\ reference resources from forms. It is hence left up to the user to insert page resources managed by us, if they need them. For that, use \pdf\pgres\() or the below \TeX\ alternative for that.

\text{pdfdict Mt = (\n  \_\_tostring = function(dict)\n  local out = {\"<<\"\n    for k, v in pairs(dict) do\n      out[#out+1] = fmt("/\%s %s", tostring(k), tostring(v))\n    end\n    out[#out+1] = \"\n  end,\n\n  local function pdf.dict(t)\n    return setmetatable(t or {}, pdfdictMt)\n  end\n  local resource_dict_objects = {}\n  local page_resources = {}\n  function pdf.add_page_resource(type, name, value)\n    local resources = page_resources[type]\n    if not resources then\n      local obj = pdf.reserveobj()\n      pdf.setpageresources(fmt("/\%s \%s 0 R", pdf.get_page_resources(), type, obj))\n      resource_dict_objects[type] = obj\n      resources = pdf.dict()\n      page_resources[type] = resources\n    end\n    page_resources[type][name] = value\n  end\n  function pdf.get_page_resources()\n    return pdf.getpageresources() or "\n  end\nend}
New “pseudo” primitives are introduced. \_addpageresource{⟨type⟩}{⟨PDF name⟩}{⟨PDF dict⟩} adds more resources of given resource ⟨type⟩ to our data structure. \_pageresources expands to the saved ⟨type⟩s and object numbers.

We write the objects with resources to the PDF file in the finish_pdffile callback.

2.39.5 Handling of colors and transparency 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.

The attribute for coloring is allocated in colors.opm

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.
The function `colorize(head, current, current_stroke, current_tr)` 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 and the current trasparency attribute. 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 or transparency 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 (−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 is_color_needed(head, n, id, subtype) -- returns fill, stroke color needed
  if id == glyph_id then
    return true, false
  elseif id == glue_id then
    n = getleader(n)
    if n then
      return true, true
    end
    return true, true
  elseif id == rule_id then
    local width, height, depth = getwhd(n)
    if width <= one_bp or height + depth <= one_bp then
      -- running (~-2^30) may need both
      return true, true
    end
    return true, false
  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, current_tr)
  for n, id, subtype in traverse(head) do
    if id == hlist_id or id == vlist_id then
      -- nested list, just recurse
      local list = getlist(n)
      list, current, current_stroke, current_tr = colorize(list, current, current_stroke, current_tr)
      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, current_tr = colorize(replace, current, current_stroke, current_tr)
        setfield(n, "replace", replace)
      else
        local fill_needed, stroke_needed = is_color_needed(head, n, id, subtype)
        local new = getattribute(n, color_attribute) or 0
        if fill_needed then
          setattribute(n, color_attribute, new + 1)
        elseif stroke_needed then
          setattribute(n, color_attribute, new - 1)
        end
      end
    elseif id == whatsit_id then
      -- this is a whatsit
      local whatsit = getfield(n, "what")
      if whatsit then
        whatsit, current, current_stroke, current_tr = colorize(whatsit, current, current_stroke, current_tr)
        setfield(n, "what", whatsit)
      end
    elseif id == text_id then
      local text = getfield(n, "text")
      if text then
        text, current, current_stroke, current_tr = colorize(text, current, current_stroke, current_tr)
        setfield(n, "text", text)
      end
    elseif id == glue_id then
      -- glue node
      local leader = getleader(n)
      if leader then
        leader, current, current_stroke, current_tr = colorize(leader, current, current_stroke, current_tr)
        setleader(n, leader)
      end
    elseif id == rule_id then
      local width, height, depth = getwhd(n)
      if width <= one_bp or height + depth <= one_bp then
        -- running (~-2^30) may need both
        return true, true
      end
      return true, false
    elseif id == glyph_id then
      -- glyph
      local glyph = getglyph(n)
      if glyph then
        glyph, current, current_stroke, current_tr = colorize(glyph, current, current_stroke, current_tr)
        setglyph(n, glyph)
      end
    elseif id == whatsit_id then
      -- whatsit
      local whatsit = getwhat(n)
      if whatsit then
        whatsit, current, current_stroke, current_tr = colorize(whatsit, current, current_stroke, current_tr)
        setwhat(n, whatsit)
      end
    end
  end
end
```
local newtr = getattribute(n, transp_attribute) or 0
local newliteral = nil
if current ~= new and fill_needed then
    newliteral = token_getmacro("_color:"..new)
current = new
end
if current_stroke ~= new and stroke_needed then
    local stroke_color = token_getmacro("_color-s:"..current)
    if stroke_color then
        if newliteral then
            newliteral = fmt("%s %s", newliteral, stroke_color)
        else
            newliteral = stroke_color
        end
        current_stroke = new
    end
end
if newtr ~= current_tr and fill_needed then -- (fill_ or stroke_needed) = fill_neded
    if newliteral ~= nil then
        newliteral = fmt("%s /tr%d gs", newliteral, newtr)
    else
        newliteral = fmt("/tr%d gs", newtr)
    end
    current_tr = newtr
end
end
if newliteral then
    head = insertbefore(head, n, pdfliteral(newliteral))
end
end
return head, current, current_stroke, current_tr
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, 0)
    return tonode(list)
end, "_colors")

We also hook into luaotfload's handling of color and transparency. Instead of the default behavior (inserting colorstack whatsis) we set our own attribute. On top of that, we take care of transparency resources ourselves.

The hook has to be registered after luaotfload is loaded.

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

local function set_node_color(n, color) -- "1 0 0 rg" or "0 g", etc.
    local attr = tonumber(token_getmacro("\_color::"..color))
    if not attr then
        attr = tex_getcount(color_count)
        tex_setcount(color_count, attr + 1)
        local strattr = tostring(attr)
        token_setmacro("\_color::"..color, strattr, "global")
        token_setmacro("\_color-s:"..strattr, color, "global")
        token_setmacro("\_color-s:"..strattr, string.upper(color), "global")
    end
    setattribute(todirect(n), color_attribute, attr)
end
optex.set_node_color = set_node_color

219
function optex.hook_into_luaotfload()
   -- color support for luaotfload v3.13+, otherwise broken
   pcall(luaotfload.set_colorhandler, function(head, n, rgbcolor) -- rgbcolor = "1 0 0 rg"
      set_node_color(n, rgbcolor)
      return head, n
   end)
   -- transparency support for luaotfload v3.22+, otherwise broken
   pcall(function()
      luatexbase.add_to_callback("luaotfload.parse_transparent", function(input) -- from "00" to "FF"
         -- in luaotfload: 0 = transparent, 255 = opaque
         -- in optex: 0 = opaque, 255 = transparent
         local alpha = tonumber(input, 16)
         if not alpha then
            tex.error("Invalid transparency specification passed to font")
            return nil
         elseif alpha == 255 then
            return nil -- this allows luaotfload to skip calling us for opaque style
         end
         local transp = 255 - alpha
         local transpv = fmt("%.3f", alpha / 255)
         pdf.add_page_resource("ExtGState", fmt("tr%d", transp), pdf_dict{ca = transpv, CA = transpv})
         pdf.add_page_resource("ExtGState", "tr0", pdf_dict{ca = 1, CA = 1})
         return transp -- will be passed to the below function
      end, "optex")
      luaotfload.set_transparenthandler(function(head, n, transp)
         setattribute(n, transp_attribute, transp)
         return head, n
      end)
   end)

   -- History:
   -- 2022-08-25 expose some useful functions in `optex` namespace
   -- 2022-08-24 luaotfload transparency with attributes added
   -- 2022-03-07 transparency in the colorize() function, current_tr added
   -- 2022-03-05 resources management added
   -- 2021-07-16 support for colors via attributes added
   -- 2020-11-11 optex.lua released

2.40 Printing documentation

The \printdoc {filename}{space} and \printdoctail {filename}{space} commands are defined after the file doc.opm is load by \load [doc].

The \printdoc starts reading of given {filename} from the second line. The file is read in the listing mode. The \printdoctail 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 \_endcode 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 the \_endcode then the \endinput 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 \`\langle sequence\rangle` in red, for example \`\langle foo\rangle`. The user documentation point is the first occurrence of \`\langle sequence\rangle`, for example \`\langle foo\rangle`. 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 \`\langle sequence\rangle` (for example \`\langle foo\rangle`) 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\`\langle foo\rangle` 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 \,\langle csname\rangle and \el:\langle csname\rangle are defined. The second macro should create a hyperlink using \_tmpa where the link name of the \langle csname\rangle is saved and \_tmpb where the name of the \langle csname\rangle to be printed is saved (\_tmpb can include preceding _ or . unlike \_tmpa). For example, suppose, that we have created \texttt{optex-doc.eref} file by:

```
TEXINPUTS='.;$TEXMF/{doc,tex}//' optex optex-doc
grep Xindex optex-doc.ref > optex-doc.eref
```

The \texttt{.eref} file includes only \_Xindex{\langle csname\rangle}{} lines from \texttt{optex-doc.ref} file. Then we can use following macros:

```
\def\_Xindex#1#2{\sdef{,#1}{}\slet{el:#1}{optexdoclink}}
\def\optexdoclink{%
  \edef\extlink{url:\optexdocurl\csstring\#cs:\_tmpa}%
  \_ea\_urlactive\_ea[\extlink]{\Cyan}{\csstring\_tmpb}}
\def\optexdocurl{http://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf}
\isfile{optex-doc.eref}\iftrue \input{optex-doc.eref}\fi
```

All \el:\langle csname\rangle, where \langle csname\rangle is from \texttt{optex-doc.ref}, have the same meaning: \optexdoclink in this example. And \optexdoclink creates the external link in \Cyan color.

### 2.40.1 Implementation

```
\_codelc1\printdoc {Macros for documentation printing <2022-12-11>} % loaded on demand by \load[doc]
```

General declarations.

```
\_fontfam[1mfonts]
```

```
\_let \mlinkcolor=\Red % main doc. points
\_let \ulinkcolor=\Blue % user doc. points
\_let \fnamecolor=\Brown % file names in listing headers
```

```
\def \bgverbcolor { \_setcmykcolor{0 0 .3 .03}} % background for listings
\def \outlinkcolor { \_setcmykcolor{1 0 1 .2}} % green for outerlinks
\def \inlinkcolor { \_setcmykcolor{0 1 0 .1}} % magenta for internal links
```

Maybe, somebody needs \seccc or \secccc?

maybe, somebody needs \seccc or \secccc?

```
\_edef\seccc#1{\_medskip \_noindent{\_bf#1}\_par \nobreak \_firstnoindent}
```

```
\_edef\secccc{\_medskip \_noindent \_bullet}
```

\enddocument can be redefined.

```
\_let\enddocument=\_bye
```

221
A full page of listing causes underfull \vbox in output routine. We need to add a small tolerance.

The listing mode is implemented here. The \maxlines is maximal lines of code printed in the listing mode. The \catcode `\.=11 sets dot as letter in listings (for package documentation where \.foo sequences 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 ⟨csname⟩ 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.
By default the internal link is created by \texttt{\_intlink} inside listing mode. But you can define \texttt{\_ea\{csname\}} which has precedence and it can create an external link. The \texttt{\_tmpa} includes the name used in the link and \texttt{\_tmpb} is the name to be printed. See \texttt{\_makecsF} above and the example at the beginning of this section.

\texttt{\_def\_intlink\{\link[cs:\_tmpa]{\ulinkcolor}{\_csstring관계\_tmpb}\}}

The lines in the listing mode have a yellow background.

\texttt{\_def\_printcodeline#1\{\advance \_maxlines by-1 \_ifnum \_maxlines<0 \_ea\_endverbprinting \_fi \_ifx \_printfilename \relax \_penalty \_ttpenalty \_fi \_vskip-4pt \_noindent \_rlap{\bgverbcolor \_vrule height8pt depth5pt width\_hsize} \_vskip-4pt \_indent \_printverblinenum \_par\}}

\texttt{\_def\_printfilename\{\hbox to0pt{\_hskip\_hsize\_vbox to0pt{\texttt\_llap{\fnamecolor\docfile}\_kern7.5pt}}\_hss} \_let \_printfilename=\relax \_everytt={\_let \_printverblinenum=\relax}}

\texttt{\_public \_printdoc \_printdoctail ;}

\texttt{\_addto \_ignoredcharsen {_} % \_foo, \_foo is the same in the fist pass of sorting \_let \_optexprintii=\printii % original \_printii used for other Index entries \_def\_printii \#1&{\_ifcsname cs:#1\_endcsname \_noindent \_hskip-\_iindent \texttt{\_link[cs:#1]{\ulinkcolor{\_csstring\_tmpb}\}} \_else \_noindent \_hskip-\_iindent \_tt \_link[cs:#1]{\ulinkcolor{\_bslash#1}} \_\else}}

\texttt{\_docfile is currently documented file. \_printdoc and \_printdoctail macros are defined here.}

You can do \texttt{\verb\put\_vitt{\_filename}} (\texttt{\_from}~\texttt{\_to}) \texttt{\_filename} if you need analogical design like in listing mode.

The Index entries are without the trailing backslash in \texttt{.ref} file. When printing Index, we distinguish the Index entries with their main documentation point (they are created as links and backslash is added), Index entries with only user documentation points have backslash added but no link is created. Other index entries are printed as usual without backslash.
If this macro is loaded by \load then we need to initialize catcodes using the \_afterload 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 \fw macro for forward links to user documentation point (given later) is defined here.
Index

There are all control sequences used in OpTEX except \TeX primitives. If you want to know something about \TeX primitives then you can use another index from \TeX in a Nutshell.

\AbbreviateFirstname 173
\aboveliskip 139
\abovetitle 133, 136
\activequotes 202
\addcitelist 163–164
\addcolor 121
\addextgstate 120, 152
\addtocorr 112
\addpagersource 152, 217
\address 25, 194–195
\addtabitemx 157
\addto 29, 40, 113
\addtomodlist 81
\addUmathfont 98, 100
\adef 17, 29, 40
\adots 93
\advancepageno 113–114
\After 173
\afterfi 29, 43
\afteritcorr 112
\afterload 55
\aheadto 29, 40, 101
\algol 27
\allocator 42
\allowbreak 61
\altquotes 201–202
\asciisortingtrue 189
\athe 140
\auescan 170
\auescanA 170
\auescanB 170
\auescanD 170
\authlist 182
\authorname 167, 173
\b 63
\backgroundbox 114
\backgroundpic 150
\balancecolumns 161
\banner 40
\basename 29, 37
\bchar 86, 104
\bbdigits 102
\bbig 91
\bbigl 91
\bbigm 91
\bbigr 91
\bbvariables 102
\bcalvariables 102
\begblock 14, 27–28, 140
\begfile 27
\begitems 13–14, 28, 51, 139
\beglua 27
\begLUA 27
\begmulti 19, 28, 51, 161
\begoutput 113, 130
\begtt 16–18, 28, 50–51, 113, 141, 144
\begti 141
\belowliskip 139
\belowtitle 133, 136
\betweencolumns 160
\bf 8–9, 69, 71, 79, 86, 104
\bfgreek 102, 105
\bfGreek 102
\bfreekvariables 102
\bfrakvariables 102
\bfvariables 102
\bgroup 39
\bi 8–9, 69, 71, 79, 86, 104
\bib 20–21, 28, 165
\bibA 165
\bibB 165
\bibentry 169
\bibgl 165
\bibmark 162, 165, 167
\bibnn 163
\bibnum 126, 162
\biboptions 52, 175
\bibp 162
\bibpart 21, 52, 162, 168
\bibskip 165
\bibtextbook 52, 166
\bibwarning 167, 173
\big 91
\Big 91
\bigbreak 61
\biggl 91
\Biggl 91
\biggm 91
\Biggm 91
\biggr 91
\Biggr 91
\bigl 91
\Bigl 91
\bigr 91
\Bigr 91
\bgroup 39
\bgroup 63
\cal 86, 104
\calvariables 102
\cancel 27
\caption 10–12, 28, 137–138
\captionF 27
\captionformat 137–138
\captionsep 137
\captionT 27
\cases 95
\casesof 29, 46–47
\catalogexclude 84
\catalogmallsample 84
\catalognextfam 84
\catalogonly 84
\catalogsample 84
\catcode 57
\cdots 92
\centerline 61
\chap 10, 12, 17–18, 28, 56, 133, 135
\chapfont 73, 133
\chapx 134