OpTEX

Format Based on Plain TEX and OPmac

Version 1.09

Petr Olšák, 2020, 2021, 2022

http://petr.olsak.net/optex

OpTEX is LuaTeX format with Plain TEX and OPmac. Only LuaTeX engine is supported.
OpTEX should be a modern Plain TEX with power from OPmac (Fonts Selection System, colors, graphics, references, hyperlinks, indexing, bibliography, ...) with preferred Unicode fonts.
The main goal of OpTEX is:
• OpTEX keeps the simplicity (like in Plain TEX and OPmac macros).
• There is no old obscurities concerning various 8-bit encodings and various engines.
• OpTEX provides a powerful Fonts Selection System (for Unicode font families, of course).
• OpTEX supports hyphenations of all languages installed in your TEX system.
• All features from OPmac macros are copied. For example sorting words in the Index, reading .bib files directly, syntax highlighting, colors, graphics, hyperlinks, references).
• Macros are documented in the same place where code is.
• User namespace of control sequences is separated from the internal namespace of OpTEX and primitives (\foo versus \_foo). The namespaces for macro writers are designed too.

If you need to customize your document or you need to use something very specific, then you can copy relevant parts of OpTEX macros into your macro file and do changes to these macros here. This is a significant difference from LaTeX or ConTeXt, which is an attempt to create a new user level with a plenty of non-primitive parameters and syntax hiding TEX internals. The macros from OpTEX are simple and straightforward because they solve only what is explicitly needed, they do not create a new user level for controlling your document. We are using TEX directly in this case. You can use OpTEX macros, understand them, and modify them.
OpTEX offers a markup language for authors of texts (like LATEX), i.e. the fixed set of tags to define the structure of the document. This markup is different from the LATEX markup. It may offer to write the source text of the document somewhat clearer and more attractive.

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

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

1 OPmac package is a set of simple additional macros to Plain TEX. It enables users to take advantage of LATEX functionality but keeps Plain TEX simplicity. See http://petr.olsak.net/opmac-e.html for more information about it.
2 All these features are implemented by TEX macros, no external program is needed.
Contents

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

2 Technical documentation ........................................ 30
  2.1 The main initialization file .................................. 30
  2.2 Concept of namespaces of control sequences ................. 32
    2.2.1 Prefixing internal control sequences ....................... 32
    2.2.2 Namespace of control sequences for users ................ 32
    2.2.3 Macro files syntax ....................................... 33
    2.2.4 Name spaces for package writers ......................... 33
    2.2.5 Summary about rules for external macro files published for OpTEX 34
2.20.2 Color mixing ........................................ 113
2.20.3 Implementation ..................................... 113
2.21 The .ref file ......................................... 118
2.22 References ........................................... 119
2.23 Hyperlinks ............................................ 121
2.24 Making table of contents .............................. 123
2.25 PDF outlines ........................................... 125
  2.25.1 Nesting PDF outlines ............................... 125
  2.25.2 Strings in PDF outlines ............................ 126
2.26 Chapters, sections, subsections ........................ 128
2.27 Lists, items ........................................... 133
2.28 Verbatim, listings ................................... 135
  2.28.1 Inline and “display” verbatim ....................... 135
  2.28.2 Listings with syntax highlighting .................. 139
2.29 Graphics .............................................. 142
2.30 The \table macro, tables and rules .................. 148
  2.30.1 The boundary declarator : ......................... 148
  2.30.2 Usage of the \tabskip primitive .................... 148
  2.30.3 Tables to given width ............................. 149
  2.30.4 eqbox: boxes with equal width across the whole document .......................... 149
  2.30.5 Implementation of the \table macro and friends ............... 150
2.31 Balanced multi-columns ................................ 155
2.32 Citations, bibliography ................................ 157
  2.32.1 Macros for citations and bibliography preloaded in the format .................. 157
  2.32.2 The \usebib command ................................ 160
  2.32.3 Notes for bib-style writers ......................... 161
  2.32.4 The usebib.om macro file loaded when \usebib is used .................. 165
  2.32.5 Usage of the bib-iso690 style .................... 165
  2.32.6 Implementation of the bib-iso690 style ............ 171
2.33 Sorting and making Index ................................ 175
2.34 Footnotes and marginal notes .......................... 182
2.35 Styles ................................................ 184
  2.35.1 \report and \letter styles ....................... 184
  2.35.2 \slides style for presentations ................... 185
2.36 Logos ................................................ 189
2.37 Multilingual support .................................. 190
  2.37.1 Lowercase, uppercase codes ....................... 190
  2.37.2 Multilingual phrases and quotation marks ........... 190
  2.37.3 Languages declaration ............................. 192
  2.37.4 Data for various languages ......................... 195
2.38 Other macros ........................................ 197
  2.39 Lua code embedded to the format ................... 199
  2.39.1 General ......................................... 199
  2.39.2 Allocators ....................................... 200
  2.39.3 Callbacks ........................................ 200
  2.39.4 Management of PDF page resources ................. 205
  2.39.5 Handling of colors and transparency using attributes ................... 206
2.40 Printing documentation ................................ 210
  2.40.1 Implementation ................................... 211

Index ................................................. 215
Chapter 1
User documentation

1.1 Starting with Op\TeX

Op\TeX{} is compiled as a format for Lua\TeX{}. Maybe there is a command \texttt{optex} in your \TeX{} distribution. Then you can write into the command line

\texttt{optex document}

You can try to process \texttt{optex op-demo} or \texttt{optex optex-doc}.

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

A minimal document should be

\begin{verbatim}
\fontfam[LMfonts]
Hello World! \bye
\end{verbatim}

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

A somewhat larger example with common settings should be:

\begin{verbatim}
\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
\end{verbatim}

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

1.2 Page layout

1.2.1 Setting the margins

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

\begin{verbatim}
\margins/{pg} \{fmt\} \{(left),\{right\},\{top\},\{bot\}\}\{unit\}
\end{verbatim}

example:

\begin{verbatim}
\margins/1 a4 (2.5,2.5,2,2)cm
\end{verbatim}

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\}.
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 $\hsize$ is set. Else $\hsize$ 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 ($\vsize$ instead $\hsize$ is used). Examples:

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

\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\times200 mm with all four margins 7 mm. The spaces before and after $\langle fmt\rangle$ parameter are necessary.
\end{verbatim}

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

1.2.2 Concept of the default page

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

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

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

\begin{verbatim}
\headline = {} \\
\footline = {\_hss\_rmfixed \_folio \_hss} % \folio expands to page number \\
\headlinedist = 14pt % from baseline of $\headline$ to top of page body \\
\footlinedist = 24pt % from last line in pagebody to baseline of $\footline$
\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{pgbackground} tokens list is empty by default but it can be used for creating a background of each page (colors, picture, watermark for example). The macro \texttt{\draft} uses this register and puts big text DRAFT as a watermark to each page. You can try it.

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

1.2.3 Footnotes and marginal notes

The Plain \TeX{}'s macro \texttt{\footnote} can be used as usual. But a new macro \texttt{\fnote\{⟨text⟩\}} is defined. The footnote mark is added automatically and it is numbered on each chapter from one\textsuperscript{2}. The ⟨text⟩ is scaled to 80 %. User can redefine footnote mark or scaling, as shown in the section 2.34.

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

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

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

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

1.3 Fonts

1.3.1 Font families

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

\textsuperscript{2} You can declare \texttt{\fnotenumglobal} if you want footnotes numbered in whole document from one or \texttt{\fnotenumpages} if you want footnotes numbered at each page from one. Default setting is \texttt{\fnotenumchapters}
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: (\caps, \cond for example). And there are four basic variant selectors (\rm, \bf, \it, \bi). The usage of variant selectors is the same as in Plain TeX: \{\it italics text\}, \{\bf bold text\} etc.

The font modifiers (\caps, \cond for example) can be used before a variant selector and they can be (independently) combined: \caps\it or \cond\caps\bf. The modifiers keep their internal setting until the group ends or until another modifier that negates the previous feature is used. So \caps \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 \currvar which does not change the selected variant but reloads the font due to (maybe newly specified) font modifier(s).

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

This is {\em important} text. % = This is {\it important\slash} text.
\it This is {\em important} text. % = This is\slash {\rm important\slash} text.
\bf This is {\em important} text. % = This is {\bi important\slash} text.
\bi This is {\em important} text. % = This is\slash {\bf important\slash} 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:

\typosize[10/12] % default of Plain TeX
\typosize[11/12.5] % font 11pt, baseline 12.5pt
\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 \typosize for example) then you must first finalize the paragraph before closing the group: {\typosize[12/14] ...{text of paragraph}... \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:

\typoscale[800/800] % fonts and baselineskip re-size to 80 %
\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{\mainfosize} and \texttt{\mainbaselineskip}.
The `\typoscale` command does scaling with respect to current values by default. If you want to do it with respect to the main values, type `\scalemain` immediately before `\typoscale` command.

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

The `\typosize` and `\typoscale` macros initialize the font family by `\rm`. You can re-size only the current font by the command `\thefontsize(⟨font-size⟩)` or the font can be rescaled by `\thefontscale(⟨factor⟩)`. These macros don’t change math fonts sizes nor baselineskip.

There is “low level” `\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 `\setfontsize{at15pt}\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 `\boldmath` variant. Most declared text font families (see `\fontfam` in the section 1.3.1) are configured with a recommended Unicode math font. This font is automatically loaded unless you specify `\noloadmath` before first `\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 `\loadmath{⟨font-file⟩}` or `\loadmath{⟨font-name⟩}` before first `\fontfam` command.

Hundreds math symbols and operators like in AMSTeX are accessible. For example `\alpha`, `\geq`, `\sum`, `\sphericalangle`, `\bumpeq`, etc. See AMSTeX manual or Typesetting Math with OpTeX for complete list of math symbols.

The following math alphabets are available:

\[\text{\texttt{\mit % mathematical variables}} \quad abc–xyz, ABC–XYZ\]
\[\text{\texttt{\it % text italics}} \quad abc–xyz, ABC–XYZ\]
\[\text{\texttt{\rm % text roman}} \quad abc–xyz, ABC–XYZ\]
\[\text{\texttt{\cal % normal calligraphics}} \quad ABC–XYZ\]
\[\text{\texttt{\script % script}} \quad A BC–X Y Z\]
\[\text{\texttt{\frak % fracture}} \quad abc–τυπ, ABC–X Y zig\]
\[\text{\texttt{\bbchar % double stroked letters}} \quad ABC–XYZ\]
\[\text{\texttt{\bf % sans serif bold}} \quad abc–xy z, ABC–XYZ\]
\[\text{\texttt{\bi % sans serif bold slanted}} \quad abc–xy z, ABC–XYZ\]

The last two selectors `\bf` and `\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 `\bf` and `\bi` selectors are documented.

The math fonts can be scaled by `\typosize` and `\typoscale` macros. Two math fonts collections are prepared: `\normalmath` for normal weight and `\boldmath` for bold. The first one is set by default, the second one is usable for math formulas in titles typeset in bold, for example.

9
You can use \textbox{$\langle$\text$\rangle$} inside math mode. It behaves as \textbox{$\{\hbox{$\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 (end of line)
\chap Chapter title (end of line)
\sec Section title (end of line)
\secc Subsection title (end of line)

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

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

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

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

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

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

1.4.2 Another numbered objects

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

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

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

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

Another automatically numbered object is a caption which is tagged by \caption/t for tables and \caption/f for figures. The caption text follows. The \cskip can be used between \caption text and the real object (table or figure). You can use two orders: \langle\caption\rangle\cskip \langle object\rangle or \langle object\rangle\cskip \langle\caption\rangle. The \cskip creates appropriate vertical space between them. Example:
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–∞</td>
<td>moderate</td>
</tr>
</tbody>
</table>

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–∞</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:

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

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\(\langle\text{letter}\rangle\{\langle\text{text}\rangle\}\) macro. For example:

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

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 $M$ 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 $\langle label \rangle$ is appended to $\backslash$chap, $\backslash$sec, $\backslash$secc, $\backslash$caption/t, $\backslash$caption/f or $\backslash$eqmark. The alternative syntax is to use $\backslash$label$\langle label \rangle$ before mentioned commands (not necessarily directly before). The reference is realized by $\backslash$ref$\langle label \rangle$ (prints the number of the referenced object) or $\backslash$pgref$\langle label \rangle$ (prints the page number). Example:

$\sec[beatle]$ About Beatles

\nindent\hfil\table{rl}{...} % the table
\cskip
\caption[t] [comp-depend] The dependency of the comp-dependency on the age.

$\label[pythagoras]$ $$ a^2 + b^2 = c^2 \eqmark $$

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

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

You can create a reference to whatever else by commands $\backslash$label$\langle label \rangle$ $\backslash$label$\langle text \rangle$. The connection between $\langle label \rangle$ and $\langle text \rangle$ is established. The $\backslash$ref$\langle label \rangle$ will print $\langle text \rangle$.

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

1.4.4 Hyperlinks, outlines

If the command $\hyperlinks \langle color-in \rangle \langle color-out \rangle$ is used at the beginning of the document, then the following objects are hyperlinked in the PDF output:

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

The last object is an external link and it is colored by $\langle color-out \rangle$. Other links are internal and they are colored by $\langle color-in \rangle$. Example:

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

You can use another marking of active links: by frames which are visible in the PDF viewer but invisible when the document is printed. The way to do it is to define the macros $\_pgborder$, $\_tocborder$, $\_citeborder$, $\_refborder$ and $\_urlborder$ 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 \langle color-fnt \rangle \langle color-fnf \rangle where footnote marks in the text have \langle color-fnt \rangle and the same footnote marks in footnotes have \langle color-fnf \rangle. 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 \langle \text{label} \rangle. The active text linked to the \dest can be created by \ilink \langle \text{label} \rangle \lbrace \langle \text{text} \rangle \rbrace. The \langle \text{type} \rangle parameter is one of the toc, pg, cite, ref, or another special for your purpose. These commands create internal links only when \hyperlinks is declared.

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

The PDF format provides outlines which are notes placed in the special frame of the PDF viewer. These notes can be managed as a structured and hyperlinked table of contents of the document. The command \outlines{\langle level \rangle} creates such outlines from data used for the table of contents in the document. The \langle level \rangle 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{\langle text \rangle} 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{\langle text \rangle} uses \langle text \rangle in the outline instead of default title text for the first following \chap, \sec, or \secc. Special case: \thisoutline\relax doesn’t create any outline for the following \chap, \sec, or \secc.

### 1.4.5 Lists

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

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

---

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

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

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

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

1.4.6 Tables
The macro \table{⟨declaration}⟨⟨data} provides similar ⟨declaration} of tables as in \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{||lc|}|{
  Month & commodity & price \crl
  January & notebook & \$ 700 \cr
  February & skateboard & \$ 100 \cr
  July & yacht & \$ 170 \crl}
generates the result:

<table>
<thead>
<tr>
<th>Month</th>
<th>commodity</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>notebook</td>
<td>$ 700</td>
</tr>
<tr>
<td>February</td>
<td>skateboard</td>
<td>$ 100</td>
</tr>
<tr>
<td>July</td>
<td>yacht</td>
<td>$ 170</td>
</tr>
</tbody>
</table>
Apart from 1, r, c declarators, you can use the \p{⟨size⟩} declarator which declares the column with paragraphs of given width. More precisely, a long text in the table cell is printed as a multiline paragraph with given width. By default, the paragraph is left-right justified. But there are alternatives:

- \p{⟨size⟩}\fL fit left, i.e. left justified, ragged right,
- \p{⟨size⟩}\fR fit right, i.e. right justified, ragged left,
- \p{⟨size⟩}\fC fit center, i.e. ragged left plus right,
- \p{⟨size⟩}\fS fit special, short one-line paragraph centered, long paragraph normal,
- \p{⟨size⟩}\fX fit extra, left-right justified but last line centered.

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

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

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

- \crl... the end of the row with a horizontal line after it.
- \crl1... the end of the row with a double horizontal line after it.
- \crlr... like \crl but the horizontal line doesn’t intersect the vertical double lines.
- \crlri... like \crlr but horizontal line is doubled.
- \crlp{⟨list⟩}... like \crl but the lines are drawn only in the columns mentioned in comma-separated ⟨list⟩ of their numbers. The ⟨list⟩ can include ⟨from⟩–⟨to⟩ declarators, for example \crlp{1-3,5} is equal to \crlp{1,2,3,5}.

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

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

\everytable={} % code used in \vbox before table processing
\thistable={} % code used in \vbox, it is removed after using it
\tabiteml={\enspace} % left material in each column
\tabitemr={\enspace} % right material in each column
\tabstrut={\strut} % strut which declares lines distance in the table
\tablinespace=2pt % additional vert. space before/after horizontal lines
\vvkern=ipt % space between lines in double vertical line
\hhkern=ipt % space between lines in double horizontal line
\tabskip=0pt % space between columns
\tabskip1=0pt \tabskipr=0pt % space before first and after last column

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

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

The \frame{⟨text⟩} makes a frame around ⟨text⟩. You can put the whole \table into \frame if you need double-ruled border of the table. Example:
First & second & third
Seven & eight & nine

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{\vrule height 20pt depth 10pt width 0pt}
  \baselineskip=20pt \tablinespace=0pt \rulewidth=.8pt}
\table{|8{c|}}{crlp{3-8}
  \mspan2[c|]{} & \mspan3[c|]{Singular} & \mspan3[c|]{Plural} \crlp{3-8}
  \mspan2[c|]{} & Neuter & Masculine & Feminine & Masculine & Feminine & Neuter \crl
  \vspan2{I} & Inclusive & \mspan3[c|]{O} & X & X & \mspan2[c|]{O} \crlp{2,4-7}
  & Exclusive & \mspan6[c|]{X} \crl
  \vspan2{II} & Informal & \mspan3[c|]{X} & X & X \crl
  & Formal & \mspan4[c|]{X} \crl
  \vspan2{III} & Informal & O & X & X & X & O \crlp{2,4-7}
  & Formal & X \crl
}
```

You can use `\vspan` with non-integer parameter too if you feel that the result looks better, for example `\vspan{2.1}{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{⟨left⟩#⟨right⟩}`. More about it is in technical documentation in section 2.30.5. See the definition of the \_tabdeclarerc 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](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. OptiTeX 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={\typosize[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 OpTEX). You can use more robust `\code{⟨text⟩}` in problematic situations, but you have to escape the following characters in the `⟨text⟩`: \, #, %, braces (if the braces are unmatched in the `⟨text⟩`), and space or ^ (if there are more than one subsequent spaces or ^ in the `⟨text⟩`). Examples:

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

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

```
\verbininput (12-42) program.c % listing from program.c, only lines 12-42
\verbininput (-60) program.c % print from begin to the line 60
\verbininput (61-) program.c % from line 61 to the end
\verbininput (-) program.c % whole file is printed
\verbininput (70+10) program.c % from line 70, only 10 lines printed
\verbininput (+10) program.c % from the last line read, print 10 lines
\verbininput (-5+7) program.c % from the last line read, skip 5, print 7
\verbininput (+) program.c % from the last line read to the end
```
You can insert additional commands for `\verbatim` before the first opening bracket. They are processed in the local group. For example, `\verbatim \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 `\verbatim` can be controlled by `\everytt, \ttindent` just like in `\begtt...\endtt`.

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

If the code is read by `\verbatim` 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 138) 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-toe⟩}{⟨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 Lo}Go}}} % used in TOC
{\def\mylogo{\hbox{{\it My}{\lo Go}}}} % used in running heads
{\def\mylogo{MyLoGo}} % used in PDF outlines

1.5.2 Making the index

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

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

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

You don’t have to double the word if you use the `\iid` instead of `\ii`:

18
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.
\iid 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 \iid parameter. The previous example is equivalent to \iid linear/space \iid 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 \iid parameter. Example:

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

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

The \ii or \iid commands can be preceded by \iitype ⟨letter⟩, then such reference (or more references generated by one \ii) has the specified type. The page numbers of such references should be formatted specially in the index. \TeX{} implements only \iitype b, \iitype i and \iitype u: the page number in bold or in italics or underlined is printed in the index when these types are used. The default index type is empty, which prints page numbers in normal font. The \TeX{}book 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. \TeX{} provides other macros \begmulti and \endmulti for more columns:

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

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

\sec Index
\begmulti 3 \makeindex \endmulti

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

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

...
The \textit{\textquotedbl} \{equivalent\} \{\textit{text}\} \textit{\textquotedbl} creates one entry in the “dictionary of the exceptions”. The sorting is done by the \{\textit{equivalent}\} \textit{\textquotedbl} but the \{\textit{text}\} \textit{\textquotedbl} 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\!Xing

The command \texttt{\cite\{\textit{label}\}} \texttt{(or \cite\{\textit{label-1},\textit{label-2},...,\textit{label-n}\})} 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 \texttt{bib}-style. For example, the citations look like [Now08] or [Nowak, 2008].

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

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

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

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

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

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

Note that another two possibilities documented in OP\!mac (using external Bib\TeX\!X program) isn’t supported because Bib\TeX\!X 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\{\textit{label}\}} command.

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

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

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

Direct reading of \texttt{.bib} files is possible by \texttt{\usebib} macro. This macro reads and uses macro package \texttt{librarian.tex} by Paul Isambert. The usage is:

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

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

Not all records are printed from \texttt{⟨bib-base⟩} 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{⟨bib-base⟩} are printed.

You can create more independent lists of references (you are creating proceedings, for example). Use \texttt{\bibpart \{⟨name⟩\}} to set the scope where \texttt{\cite}s 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[label1X] ... \cite[label1Y] ... % They belong to AA bib-list
\usebib/c (simple) file.bib % generates AA bib-list numbered 1, 2, ...
% \cite prints [1], [2], ... by bib-list AA
\bibpart {BB}
...\cite[label2Z] ... \cite[label1X] ... % They belong to BB bib-list
\bibnum=0 \usebib/c (simple) my.bib % generates BB bib-list numbered 1, 2, ...
% \cite prints [1], [2], ... by bib-list BB
\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{op tex} 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 \textsc{btp x} file \texttt{x11nam.def}. There are about 300 color names like \texttt{\deeppink}, \texttt{\chocol ate} etc. If there are numbered variants of the same name, then the letters B, C, etc. are appended to the name in \textsc{op tex}. For example \texttt{\chocolate} is Chocolate1, \texttt{\chocolateB} is 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 \texttt{\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 \texttt{\onlyrgb} creates colors more bright (usable for computer presentations). On the other hand, CMYK makes colors more true\textsuperscript{5} for printing.

You can define your color by a linear combination of previously defined colors using \texttt{\colordef}. For example:

\begin{verbatim}
Printed output is more equal to the monitor preview especially if you are using ICC profile for your printer.
\end{verbatim}
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\{\text\}

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

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 \langle img\rangle subdirectory. Note: the directory name must end by / in the \picdir declaration.

Inkscape\footnote{A powerful and free Wysiwyg editor for creating vector graphics.} is able to save a picture to PDF and labels of the picture to another file\footnote{Chose “Omit text in PDF and create LaTeX file” option.}. This second file should be read by \TeX to print labels in the same font as document font. Op\TeX supports this feature by \linkinspic \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 and include it by \inspic. If you want to “program” such pictures then Tikz package is recommended. It works in Plain \TeX and Op\TeX.

1.6.3 PDF transformations

All typesetting elements are transformed by linear transformation given by the current transformation matrix. The \pdfsetmatrix \{\langle a\rangle \langle b\rangle \langle c\rangle \langle d\rangle\} 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 \([\langle a \rangle, \langle b \rangle]\) and \([1, 0]\) to \([\langle c \rangle, \langle d \rangle]\). The stack-oriented commands \pdfsave and \pdfrestore gives a possibility of storing and restoring the current transformation matrix and the position of the current point. This position has to be the same from \TeX’s point of view as from the transformation point of view when \pdfrestore is processed. Due to this fact the \pdfsave\rlap{(transformed text)}\pdfrestore or something similar is recommended.

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

\begin{verbatim}
\pdfscale{⟨horizontal-factor⟩}{⟨vertical-factor⟩}
\pdfrotate{⟨angle-in-degrees⟩}
\end{verbatim}

These macros simply call the properly \pdfsetmatrix command.

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

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

This gives the following result. First: 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 ⟨transformation⟩ parameter includes one or more transformation commands \pdfsetmatrix, \pdfscale, \pdfrotate with their parameters. The ⟨text⟩ is transformed text.

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

The \rotbox{⟨deg⟩}{⟨text⟩} is shortcut for \transformbox{\pdfrotate{⟨deg⟩}}{⟨text⟩}.

1.6.4 Ovals, circles

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

\begin{verbatim}
\ovalparams={\roundness=2pt % diameter of circles in the corners
\fcolor=Yellow % color used for filling oval
\lcolor=Red % line color used in the border
\width=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
\end{verbatim}

The total distance from text to oval boundary is \hhkern\roundness at the left and right sides and \vvkern\roundness at the top and bottom sides of the text.
If you need to set a parameters for the \text (color, size, font etc.), put such setting right in front of the \text: \inval\{text settings\}{text}.

The \incircle\{ratio=1\}\{text\} creates a box like this \text. The \ratio parameter means width/height. The usage is analogous like for oval. The default parameters are

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

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

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

1.6.5 Putting images and texts wherever

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

The \putpic\{x\}\{y\}\{width\}\{height\}\{image-file\} puts an image given by \image-file (including extension) of given \width and \height at given position (its left-bottom corner). You can write \nospec instead \width or \height 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 \langid\lang\command in your document. For example \delang\lang for German, \cslang\lang for Czech, \pllang\lang for Polish. The \langid\lang is a shortcut of the language (mostly from ISO 639-1). You can list all available languages including their \langlist parameter is case insensitive. More exactly the left edge of its baseline is at that position.

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

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

You can switch between language patterns by \langid\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 \iso. OpTeX declares these words
only for few languages: Czech, German, Spanish, French, Greek, Italian, Polish, Russian, Slovak
and English. If you need to use these words in other languages or you want to auto-generate
more words in your macros, then you can declare it by \def or \_langw commands as shown
in section 2.37.2.

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

If you declare \langle iso-code \rangle quotes, 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. Op\TeX declares quotes only for Czech,
German, Spanish, French, Greek, Italian, Polish, Russian, Slovak and English (\csquotes,
\dequotes,..., \enquotes). You can simply define your own quotes as shown in section 2.37.2.
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 \rangle quotes\altquotes in such case.

1.7.2 Pre-defined styles

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

The \report declaration is intended to create reports. It sets default font size to 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\langle end-line\rangle 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
eamples. 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\ldots to put the address box to the right side of the document. Or you can use
\langle prefixed text\rangle\address\ldots to put \langle prefixed text\rangle before the first line of the address.

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

Analogical declaration macro \book is not prepared. Each book needs individual typo-
graphical care. You need to create specific macros for design.
1.7.3 Loading other macro packages

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

\load [qrcode, scanbase]

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

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

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

- math.opm provides usable features for math typesetting and shows how to create new packages.
- qrcode.opm enables to create QR codes.
- tikz.opm does \input tikz.tex, i.e. loads TikZ. It adds OpTEX-specific code.
- mte.opm includes settings for microtypographic extensions (protrusions+expanding fonts).
- vlna.opm enables to protect of one-letter prepositions and more things automatically.
- emoji.opm defines \emoji{⟨name⟩} command for colored emoticons.
- minim-mp.opm enables \directmetapost using minim-mp and minim packages.
- pdfextra.opm allows the use of many extra features from PDF standard (by M. Vlasák).

See these files in optex/pkg/ or optex/⟨pkgname⟩ for more information about them. The packages may have their documentation, try 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 \lorem{⟨number⟩} or \lorem{⟨from⟩ ⟨to⟩}. It prints a paragraph (or paragraphs) with neutral text. The numbers ⟨number⟩ or ⟨from⟩, ⟨to⟩ must be in the range 1 to 150 because there are 150 paragraphs with neutral text prepared for you. The \lipsum macro is equivalent to \lorem. Example: \lipsum[1-150] prints all prepared paragraphs.

If the dot follows the argument before closing ] (for example \lipsum[3.]) 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:

We are using \TeX/ because it is cool. \OpTeX/ is better than \LaTeX.

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 OpTeX

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

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

1.8 Summary

\texttt{\tit Title (terminated by end of line)}
\texttt{\chap Chapter Title (terminated by end of line)}
\texttt{\sec Section Title (terminated by end of line)}
\texttt{\secc Subsection Title (terminated by end of line)}
\texttt{\maketoc % table of contents generation}
\texttt{\ii item1,item2 % insertion the items to the index}
\texttt{\makeindex % the index is generated}
\texttt{\label [labname] % link target location}
\texttt{\ref [labname] % link to the chapter, section, subsection, equation}
\texttt{\pgref [labname] % link to the page of the chapter, section, ...}
\texttt{\caption/t % a numbered table caption}
\texttt{\caption/f % a numbered caption for the picture}
\texttt{\eqmark % a numbered equation}
\texttt{\begitems % start a list of the items}
\texttt{\enditems % end of list of the items}
\texttt{\begblock % start a block of text}
\texttt{\endblock % end of block of text}
\texttt{\begtt % start a verbatim text}
\texttt{\endtt % end verbatim text}
\texttt{\verbchar X % initialization character X for in-text verbatim}
\texttt{\code % another alternative for in-text verbatim}
\texttt{\verbinput % verbatim extract from the external file}
\texttt{\begmulti num % start multicolumn text (num columns)}
\texttt{\endmulti % end multicolumn text}
\texttt{\cite [labnames] % refers to the item in the lits of references}
\texttt{\rcite [labnames] % similar to \cite but [] are not printed.}
\texttt{\sortcitations \shortcitations \nonumcitations % cite format}
\texttt{\bib [labname] % an item in the list of references}
\texttt{\usebib/? (style) bib-base % direct using of .bib file, ? in \{s,c\}}
\texttt{\load [filenames] % loading macro files}
\texttt{\fontfam [FamilyName] % selection of font family}
\texttt{\typosize [font-size/baselineskip] % size setting of typesetting}
\texttt{\typoscale [factor-font/factor-baselineskip] % size scaling}
\texttt{\thefontsize [size] \thefontscale [factor] % current font size}
\texttt{\inspic file.ext % insert a picture, extensions: jpg, png, pdf}
\texttt{\table {rule}{data} % macro for the tables like in LaTeX}
\texttt{\fnote {text} % footnote (local numbering on each page)}
\texttt{\mnote {text} % note in the margin (left or right by page number)}
\texttt{\hyperlinks {color-in}{color-out} % PDF links activate as clickable}
\texttt{\outlines {level} % PDF will have a table of contents in the left tab}
\texttt{\magscale [factor] % resize typesetting, line/page breaking unchanged}
\texttt{\margins/pg format (left, right, top, bottom)unit % margins setting}
\texttt{\report \letter \slides % style declaration macros}
1.9 API for macro writers

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

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

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

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

\texttt{\_addto \_macro{(text)}}\ \texttt{\_addto} adds \texttt{(text)} at the end of \texttt{\_macro} body. \texttt{\_addto \_macro{(text)}} puts \texttt{(text)} at the begin. \texttt{\_addto \_macro{(text)}} adds \texttt{(text)} to \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{\_\_addto} macro below, so this macro is not available as \texttt{\_addto}.

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

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

\texttt{\addto \_macro{(text)}} \texttt{\addto} adds \texttt{(text)} at the end of \texttt{\_macro} body, \texttt{\_addto \_macro{(text)}} puts \texttt{(text)} at the begin. \texttt{\_addto \_macro{(text)}} adds \texttt{(text)} to \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{\_\_addto} macro below, so this macro is not available as \texttt{\_addto}.

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

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

\texttt{\addto \_macro{(text)}} \texttt{\addto} adds \texttt{(text)} at the end of \texttt{\_macro} body, \texttt{\_addto \_macro{(text)}} puts \texttt{(text)} at the begin. \texttt{\_addto \_macro{(text)}} adds \texttt{(text)} to \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{\_\_addto} macro below, so this macro is not available as \texttt{\_addto}.

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

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

\texttt{\addto \_macro{(text)}} \texttt{\addto} adds \texttt{(text)} at the end of \texttt{\_macro} body, \texttt{\_addto \_macro{(text)}} puts \texttt{(text)} at the begin. \texttt{\_addto \_macro{(text)}} adds \texttt{(text)} to \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{\_\_addto} macro below, so this macro is not available as \texttt{\_addto}.

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

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

\texttt{\addto \_macro{(text)}} \texttt{\addto} adds \texttt{(text)} at the end of \texttt{\_macro} body, \texttt{\_addto \_macro{(text)}} puts \texttt{(text)} at the begin. \texttt{\_addto \_macro{(text)}} adds \texttt{(text)} to \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{\_\_addto} macro below, so this macro is not available as \texttt{\_addto}.

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

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

\texttt{\addto \_macro{(text)}} \texttt{\addto} adds \texttt{(text)} at the end of \texttt{\_macro} body, \texttt{\_addto \_macro{(text)}} puts \texttt{(text)} at the begin. \texttt{\_addto \_macro{(text)}} adds \texttt{(text)} to \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{\_\_addto} macro below, so this macro is not available as \texttt{\_addto}.

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

The alphabetically sorted list of macros typically usable for macro writers follows. More information about such macros can be found in the technical documentation. You can use hyperlinks here in order to go to the appropriate place of the technical documentation.
\setctable and \restorectable manipulate with stack of catcode tables.
\slet{⟨stringA⟩}{⟨stringB⟩} behaves like \let{⟨stringA⟩}{⟨stringB⟩}
\sdef{⟨string⟩}{⟨parameters⟩}{⟨body⟩} behaves like \xdef{⟨string⟩}{⟨parameters⟩}{⟨body⟩}.
\trycs{⟨string⟩}{⟨text⟩} expands ⟨string⟩ if it is defined else expands ⟨text⟩.
\useit{⟨one⟩}, \usesecond{⟨one⟩}{⟨two⟩} uses given parameter.
\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 T\TeX

All macros of Plain T\TeX are re-written in Op\TeX. Common macros should work in the same sense as in original Plain T\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 T\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 users document too). Users should re-define unprefixed forms of control sequences without worries that something internal will be broken.

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

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

The text accents macros \", \', \v, \u, \=, \^, \., \H, \~, \` 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 T\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 T\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 T\TeX declares this control sequence for math secant\footnote{Use $\sec(x)$ to get sec($x$).}.

1.11 Related documents

• Typesetting math with Op\TeX – More details about math typesetting.
• T\TeX in a Nutshell – Summary about T\TeX principles, T\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{The math accents macros like \acute, \bar, \dot, \hat still work.}
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

```
luatex -ini optex.ini
```

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

```
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 \TeX{} 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 `\^^I=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 `\~b=12 % normal character
\def\optexversion{1.09 Dec 2022}
\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.

```
\def\optexversion{1.09 Dec 2022}
\def\fmtname{Optex}
\let\fmtversion=\optexversion
```

We check if Lua\TeX{} engine is used at \ini state. And the ``` character is set as `\newlinechar`. 30
% Engine testing:
\newline\char=`\^^J
\ifx\directlua\undefined
\message{This format is based only on LuaTeX, use luatex -ini optex.ini\^^J}
d\endinput \fi
\ifx\bgroup\undefined \else
\message{This file can be used only for format initialisation, use luatex -ini\^^J}
\endinput \fi

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

% Basic .opm syntax:
\let\_endcode =\endinput
\def \_codedecl #1#2\{\immediate\write-1{#2}\} % information about .opm file
\long\def \_doc#1\_cod#2 {} % skip documentation

Individual *.opm macro files are read.

\message{OpTeX (Olsak's Plain TeX) initialization \optexversion\^^J}
\input prefixed.opm % prefixed primitives and code syntax
\input luatex-ini.opm % LuaTeX initialization
\input basic-macros.opm % basic macros
\input alloc.opm % allocators for registers
\input if-macros.opm % special \if-macros, \is-macros and loops
\input parameters.opm % parameters setting
\input more-macros.opm % OpTeX useful macros (todo: doc)
\input keyval.opm % key=value dictionaries
\input plain-macros.opm % plainTeX macros
\input fonts-preload.opm % preloaded Latin Modern fonts
\input fonts-resize.opm % font resizing (low-level macros)
\input fonts-select.opm % font selection system
\input math-preload.opm % math fans CM + AMS preloaded
\input math-macros.opm % basic macros for math plus mathchardefs
\input unimath-macros.opm % macros for loading UnicodeMath fonts
\input fonts-opmac.opm % font managing macros from OPmac
\input output.opm % output routine
\input margins.opm % macros for margins setting
\input colors.opm % colors
\input ref-file.opm % ref file
\input references.opm % references
\input hyperlinks.opm % hyperlinks
\input maketoc.opm % maketoc
\input outlines.opm % PDF outlines
\input pdfuni-string.opm % PDFunicode strings for outlines
\input sections.opm % titles, chapters, sections
\input lists.opm % lists, \begitems, \enditems
\input verbatim.opm % verbatim
\input hi-syntax.opm % syntax highlighting of verbatim listings
\input graphics.opm % graphics
\input table.opm % table macro
\input multicolcolumns.opm % more columns by \begmulti ...\endmulti
\input cite-bib.opm % Bibliography, \cite
\input makeindex.opm % Make index and sorting
\input fnotes.opm % \fnotes, \mnotes
\input styles.opm % styles \report, \letter
\input logos.opm % standard logos
\input uni-lcuc.opm % Setting lccodes and uccodes for Unicode characters
\input languages.opm % Languages macros
\input lang-decl.opm % Languages declaration
\input others.opm % miscellaneous

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

2.2 Concept of namespaces of control sequences

2.2.1 Prefixing internal control sequences

All control sequences used in \OPMTEX are used and defined with _ prefix. The user can be sure that when he/she does \def\foo then neither internal macros of \OPMTEX nor \TEX primitives will be damaged. For example \def\if{...} will not damage macros because \OPMTEX’s macros are using _\if instead of \if.

All \TEX primitives are initialized with two representative control sequences: \word and _\word, for example \hbox and _\hbox. The first alternative is reserved for users or such control sequences can be re-defined by a user.

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

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

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

\public ⟨list of control sequences⟩;

For example \public \foo \bar ; does \let\foo=_\foo, \let\bar=_\bar.

At the end of each code segment in \OPMTEX, the _\public macro is used. You can see which macros are defined for public usage in that code segment.

The macro \private does the reverse job of \public with the same syntax. For example \private \foo \bar ; does \let\foo=\_\foo, \let\bar=\_\bar. This should be used when an unprefix 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 _\public (or \public) macro is used.

2.2.2 Namespace of control sequences for users

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

User can define or declare control sequences with _ character, for example \my_control_sequence, but with the following exceptions:

- Control sequences which begin with _ are reserved for \TEX primitives, \OPMTEX internal macros and packages internal macros.
- Multiletter control sequences in the form \⟨word⟩_ or \langle word⟩_⟨one-letter⟩, where ⟨word⟩ is a sequence of letters, are inaccessible, because they are interpreted as \⟨word⟩ followed by _ or as \langle word⟩ followed by _⟨one-letter⟩. This is important for writing math, 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}

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

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

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

2.2.3 Macro files syntax

Segments of OMap macros or external macro packages are stored in files with .opm extension (means OMap 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 {⟨short title⟩ ⟨version⟩}

If the mentioned \sequence is undefined then \codedecl prints the message

\%: [[file name]] ⟨short title⟩ ⟨version⟩

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 \load[doc] 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 documentation between first pair of \doc ... \cod used after \endcode. These macros should \load[doc] and must be finished by \bye. Then you have code+documentation together in a single file and user can generate the documentation of our package by \docgen used at command line:

\texttt{optex -jobname pkgname-doc '\docgen pkgname'}

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

2.2.4 Name spaces for package writers

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

The package writer does not need to write repeatedly \pkg{foo} \pkg{bar} etc. again and again in the macro file.\footnote{We have not adopted the idea from expl3 language:} When the \namespace{⟨pkg⟩} is declared at the beginning of the macro file then all occurrences of \foo will be replaced by \namespace{⟨pkg⟩}foo at the input processor level. The macro writer can write (and backward can read his/her code) simply with \foo, \bar control sequences and \namespace{⟨pkg⟩}foo, \namespace{⟨pkg⟩}bar control sequences are processed internally. The scope of the \namespace command ends at the \endcode command or when another \namespace is used. This command checks if the same package label is not declared by the \namespace twice.

\nspublic \listofsequences; \let\foo = \namespace{⟨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 user name space by \nspublic \macro;
It could happen that a package writer needs to declare a control sequence (say \foo) directly without setting it in \_{\langle pkg\rangle} \_foo name space followed by using \_nspublic. The \newpublic prefix should be used in this case, for example \newpublic \_def \foo or \newpublic \_chardef \foo or \newpublic \_long \_def \foo. The \newpublic \langle do \rangle \langle sequence \rangle prints a warning if the declared \langle sequence \rangle is defined already and then runs \langle do \rangle \langle sequence \rangle. 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 \{ \langle pkg\rangle \} is initialized. Or close your namespace by \endnamespace and open it again (after other packages are loaded) by \resetnamespace \{ \langle pkg\rangle \}.

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 \{ \langle pkg\rangle \} \_bar, for example \newif \_ifqr_incorner. Then the control sequences \_qr_incornertrue and \_qr_incornerfalse can be used (or the sequences \_incornertrue and \_incornerfalse when \namespace \{ qr \} is used).

2.2.5 Summary about rules for external macro files published for OpTEX

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

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

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

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

2.2.6 The implementation of the namespaces and macros for macro-files

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

\ea is useful shortcut for \expandafter. We recommend to use always the private form of \_ea because there is high probability that \ea will be redefined by the user.

\public \langle sequence \rangle \langle sequence \rangle ... ; does \let \langle sequence \rangle = \_\langle sequence \rangle for all sequences.
\private ⟨sequence⟩ ⟨sequence⟩ ...; does \let \langle sequence⟩ = \langle sequence⟩ for all sequences. \newpublic⟨do⟩⟨sequence⟩ prints warning if ⟨sequence⟩ is declared already. Then runs ⟨do⟩⟨sequence⟩.

\_checkexists ⟨where⟩ {⟨sequence-string⟩} prints error if the control sequence given by its name ⟨sequence-string⟩ is not declared. This check is used in \public, \private, \_nspublic and \_nsprivate macros in order to avoid mistakes in names when declaring new control sequences.

\xargs ⟨what⟩ ⟨sequence⟩ ⟨sequence⟩ ...; does ⟨what⟩⟨sequence⟩ for each sequences.

We define the macros \_namespace {⟨pkg label⟩}, \_resetnamespace {⟨pkg label⟩}, \_endnamespace, \_pkglabel, \_nspublic, and \_nsprivate for package writers, see section 2.2.4.

\_let\_ea = \_expandafter % usefull shortcut
\_long\_def \_xargs #1#2{% \_ifx #2; \_else \_ea#1\_ea#2\_ea\_xargs \_ea \_fi}
\_def \_pkglabel{}% \_def \_nspublic A #1{% \_checkexists \_nspublic \_csstring #1% \_ea \_let \_csname \_csstring #1\_endcsname =#1%}
\_def \_nsprivate A #1{% \_checkexists \_nsprivate \_csstring #1% \_ea \_let \_csname \_csstring #1\_endcsname =#1%}
\_def \_endnamespace {% \_directlua{ callback.remove_from_callback("process_input_buffer", \_namespace") %}
\_def \_nspublic A #1{% \_checkexists \_nspublic \_csstring #1% \_ea \_let \_csname \_csstring #1\_endcsname =#1%}
\_def \_nsprivate A #1{% \_checkexists \_nsprivate \_csstring #1% \_ea \_let \_csname \_csstring #1\_endcsname =#1%}
Each macro file should begin with \_codedecl \macro {\langle info\rangle}. If the \macro is defined already then the \endinput protects to read such file more than once. Else the \langle info\rangle is printed to the terminal and the file is read. The \_endcode is defined as \endinput in the optex.ini file. \wterm {\langle text\rangle} prints the \langle text\rangle to the terminal and to the .log file, \wlog {\langle text\rangle} prints the \langle text\rangle only to the .log file (as in plain \TeX).
2.4 Basic macros

We define first bundle of basic macros.

\begin{verbatim}
_protected_edef \pdffontname { \_pdfvariable fontname} % preloaded in format
_edef \pdffontname { \_csstring pdffont}

\let \_egroup= \let \_bgroup=
_edef \_null { \_hbox{}}
_edef \_space { }
_let \_bgroup=( \_let \_egroup=)
_let \_empty {} \_def \_empty {} \_def \_space {} \_def \_null {} \_def \_hbox {} \_def \empty {}
_let \_parens= ( \_let \_brackets= [ \_let \_keywords= ] \_let \_list= , \_let \_tildes= ~ \_let \_csstring= \_csstring}

\_protected_edef \_bslash { \_csstring\} \_protected_edef \_nbb { \_bslash \_bslash} \_protected_edef \_pcent { \_csstring \%}
\_public \bslash \nbb \pcent ;
\end{verbatim}

\sdef {⟨text⟩} is equivalent to \def⟨text⟩, where ⟨text⟩ is a control sequence. You can use arbitrary parameter mask after \sdef⟨text⟩, don’t put the (unwanted) space immediately after closing brace. \sxdef {⟨text⟩} is equivalent to \xdef⟨text⟩.
\slet ⟨textA⟩{⟨textB⟩} is equivalent to \let \langle textA \rangle = \langle textB \rangle.

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

\sdef {⟨text⟩} is equivalent to \def⟨text⟩, where ⟨text⟩ is a control sequence.
\def \{char\}\{body\} puts the \emph{char} as active character and defines it as \{\emph{body}\}. You can declare a macro with parameters too. For example \def @#1{...#1...}.

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

\trycs \{csname\}\{text\} expands to \csname \{text\}\endcsname if it is defined else to the \langle\text\rangle.

\addto \macro\{text\} adds \langle\text\rangle to your \macro, which must be defined. \aheadto \macro\{text\} defines \macro as \langle\text\rangle followed by the original \macro body.

\incr \langle\counter\rangle increases \langle\counter\rangle by one globally. \decr \langle\counter\rangle decreases \langle\counter\rangle by one globally.

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

\loggingall and \tracingall are defined similarly as in plain \TeX, but they print more logging information to the log file and the terminal.

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 \everyjob and in \docgen

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

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

Unlike in PlainT\TeX, 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\TeX, 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.

\begin{verbatim}
\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
}
\end{verbatim}

The limits are set first.

Each allocation macro needs its own counter.

\begin{verbatim}
\_countdef \_countalloc=10 \_countalloc=255
\_countdef \_dimenalloc=11 \_dimenalloc=255
\_countdef \_skipalloc=12 \_skipalloc=255
\_countdef \_muskipalloc=13 \_muskipalloc=255
\_countdef \_boxalloc=14 \_boxalloc=255
\_countdef \_toksalloc=15 \_toksalloc=255
\_countdef \_readalloc=16 \_readalloc=1
\_countdef \_writealloc=17 \_writealloc=0 % should be -1 but there is bug in new luatex
\_countdef \_famalloc=18 \_famalloc=42 % \newfam are 43, 44, 45, ...
\_countdef \_languagealloc=19 \_languagealloc=0
\end{verbatim}

The common allocation macro \texttt{\_allocator \langle sequence\rangle \{\langle type\rangle\} \langle primitive declarator\rangle} is defined. This idea was used in classical plain T\TeX by Donald Knuth too but the macro from plain T\TeX seems to be more complicated.)

\begin{verbatim}
\_def\_allocator #1#2#3{%
  \_incr{\_cs{\_#2alloc}}%
  \_ifnum\_cs{\_#2alloc} > \_cs{\_mai#2}%
    \errmessage{No room for a new \texttt{\String}\texttt{\count}\langle\textit{type}\rangle}\%
  \else
    \_global\chardef#1=\_cs{\_#2alloc}
  \fi
}
\end{verbatim}
The allocation macros \newcount, \newdimen, \newskip, \newmuskip, \newtoks, \newline read, \newwrite, \newfam, and \newlanguage are defined here.

\newcount #1{\allocator #1{count}\countdef}
\newdimen #1{\allocator #1{dimen}\dimendef}
\newskip #1{\allocator #1{skip}\skipdef}
\newmuskip #1{\allocator #1{muskip}\muskipdef}
\newbox #1{\allocator #1{box}\chardef}
\newtoks #1{\allocator #1{toks}\toksdef}
\newread #1{\allocator #1{read}\chardef}
\newwrite #1{\allocator #1{write}\chardef}
\newfam #1{\allocator #1{fam}\chardef}
\newlanguage #1{\allocator #1{language}\chardef}

\newinsert macro is defined differently than others.
\newcount _insertalloc = 255
\newdimen _insertmin = 201
\newcount _marksalloc = 0
\newcount _attributealloc = 0
\newcount _catcodetablealloc = 10
\newdimen _maxdimen = 16383.99999pt
\newdimen _zo = 0pt
\newdimen _hideskip = -1000pt
\newdimen _centering = 0pt
\newdimen _zoskip = 0pt
\newbox _voidbox

We declare public and private versions of \tmpnum and \tmpdim registers separately. They are independent registers.

A few registers are initialized like in plain\TeX. We absolutely don’t support the \@category dance, so \z@skip, \p@, etc. are defined but not recommended in Op\TeX.

The \zo and \zoskip (equivalents to \z@ and \z@skip) are declared here and used in some internal macros of Op\TeX for improving speed.
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 \if\xxx you can use \xxxtrue or \xxxfalse to set the boolean value and use \if\xxx 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).

\begin{verbatim}
def_newifi #1{\ea_newifiA \string #1\relax#1}
def \_newifiA \string\if #1\relax#2{% sdef{_#1true}{\let#2=\iftrue} sdef{_#1false}{\let#2=\iffalse} let#2=\iffalse }
def \_newifi #1{\ea_newifiA \string#1\relax#1}
def \_if #1\relax#2{% sdef{_#1true}{\let#2=\iftrue} sdef{_#1false}{\let#2=\iffalse} let#2=\iffalse }
enddef
\public \newif ;
\end{verbatim}

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

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

Nested \if ..., \afterfi{\if...\afterfi...\fi} \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 \loop are possible only in a \TeX group.

\begin{verbatim}
def \_loop #1\repeat{\def\_body{#1}\_iterate}
def \loop #1\repeat{\def\_body{#1}\_iterate}
def \_iterate \{\_body \ea \_iterate \fi}
def \_iterate \{\_body \ea \_iterate \fi}
\end{verbatim}

\foreach ⟨list⟩\do {⟨what⟩} repeats ⟨what⟩ for each element of the ⟨list⟩. The ⟨what⟩ can include #1 which is substituted by each element of the ⟨list⟩. The macro is expandable.

\foreach ⟨list⟩\do ⟨parameter-mask⟩{⟨what⟩} reads parameters from ⟨list⟩ repeatedly and does ⟨what⟩ for each such reading. The parameters are declared by ⟨parameter-mask⟩. Examples:

\begin{verbatim}
\foreach (a,1)(b,2)(c,3)\do (#1,#2){#1=#2} \foreach word1,word2,word3,\do #1,\{Word is #1.\}
\foreach A=word1 B=word2 \do #1=#2 {"#1 is set as #2".}
\end{verbatim}

Note that \foreach ⟨list⟩\do {⟨what⟩} is equivalent to \foreach ⟨list⟩\do #1{⟨what⟩}.

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.

42
The \fornum (from) .. (to) \do {⟨what⟩} or \fornumstep ⟨num⟩: (from) .. (to) \do {⟨what⟩} repeats ⟨what⟩ for each number from ⟨from⟩ to ⟨to⟩ (with step ⟨num⟩ or with step one). The ⟨what⟩ can include #1 which is substituted by current number. The ⟨from⟩, ⟨to⟩, ⟨step⟩ parameters can be numeric expressions.

The macro is expandable.

The test in the \_fornumB says: if ⟨⟨to⟩⟩ < ⟨current number⟩ AND ⟨⟨step⟩⟩ is positive) or if ⟨⟨to⟩⟩ > ⟨⟨current number⟩⟩ AND ⟨⟨step⟩⟩ is negative) then close loop by \_getforstack. Sorry, the condition is written by somewhat cryptoid TEX language.

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 ABC \do {\fornum 1..5 \do {letter: #1, number: ##1. }}

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 \for* is initialized and \_getforstack is used when the \for* macro ends. The \_forlevel variable keeps the current nesting level. If it is zero, then we need not save nor restore any data.
User can define own expandable “foreach” macro by \foreachdef \macro \langle parameter-mask \rangle \{ \langle what \rangle \} which can be used by \macro \{ \langle list \rangle \}. The macro reads repeatedly parameters from \langle list \rangle using \langle parameter-mask \rangle and does \langle what \rangle for each such reading. For example

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

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

\begin{verbatim}
\_def \_foreachdef#1#2#{\_toks0{#2}%
\_long\edef#1##1{\_ea\_noexpand\_csname _body:\_csstring#1\_endcsname
\_noexpand\_finbody}%
\_foreachdefA#1{#2}}
\_long\def\_foreachdefA#1#2#3{%
\_long\sdef{\_body:\_csstring#1}#2{\_testparam##1..\_iftrue #3\_cs{\_body:\_csstring#1\_ea}\_fi}}
\public \foreachdef ;
\end{verbatim}

2.6.3 Is-macros

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

```
\issething \langle params \rangle \iftrue \langle codeA \rangle \else \langle codeB \rangle \fi
or
\issething \langle params \rangle \iffalse \langle codeB \rangle \else \langle codeA \rangle \fi
```

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

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

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

\ isempty \{ \langle text \rangle \}\iftrue is true if the \langle text \rangle is empty. This macro is expandable.

\ istoksempty \{ \langle tokens variable \rangle \}\iftrue is true if the \langle tokens variable \rangle is empty. It is expandable.

\ isequal \{ \langle textA \rangle \} \{ \langle textB \rangle \}\iftrue is true if the \langle textA \rangle and \langle textB \rangle are equal, only from strings point of view, category codes are ignored. The macro is expandable.

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

\ isdefined \{ \langle csname \rangle \}\iftrue is true if \langle csname \rangle is defined. The macro is expandable.

\ isinlist \list\{ \langle text \rangle \}\iftrue is true if the \langle text \rangle is included the macro body of the \list. The category codes are relevant here. The macro is expandable.
\isfile{(filename)} \iftrue is true if the file (filename) exists and are readable by \TeX.

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

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

\casesof ⟨token⟩ ⟨list of cases⟩ implements something similar to the switch command known from C language. It is expandable macro. The ⟨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 _ifx primitive. The parts ⟨what to do⟩ can be finalized by a macro which can read more data from the input stream as its parameters.

_\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} {equal to zero}
_\finc {negative} value
The `\texttt{casesof}` macro works with principle: first true condition wins, next conditions are not evaluated.

```latex
\texttt{\_long\_def \_xcasesof \#1\{\_xcasesofA \#1\_finc}
\texttt{\_long\_def \_xcasesofA \#1\#2\_finc \#3\{%
\_ifx \#1\_finc \_ea\_ignoresecond\_else \_ea\_usesecond\_fi
\#3\{\_finc\#3\}\_finc\}
\texttt{\_finc\}
\texttt{\_xcasesof\}}%
\texttt{\_public \texttt{\_xcasesof} ;}
```

### 2.7 Setting parameters

The behavior of document processing by OpTeX is controlled by parameters. The parameters are

- primitive registers used in build-in algorithms of \TeX,  
- registers declared and used by Op\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. Op\TeX\ declares two control sequences for each register: prefixed (private) and unprefixed (public). Op\TeX\ macros use only prefixed variants of control sequences. The user should use the unprefixed variant with the same meaning and set or read the values of registers using the unprefixed variant. If the user re-defines the unprefixed control sequence of a register then Op\TeX\ macros still work without change.

```latex
\texttt{\_ codedecl \normalbaselineskip \{Parameter settings <2021-04-13>} % preloaded in format
```

#### 2.7.1 Primitive registers

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

```latex
\texttt{\_parindent=20pt % indentation of paragraphs
\_pretolerance=100 % parameters used in paragraph breaking algorithm
\_tolerance=200
\_hbadness=1000
\_vbadness=1000
\_doublehyphenpenalty=10000
\_finalhyphenpenalty=5000
\_adjdemerits=10000
\_uchyph=1
\_defaultthyphenchar='\-
\_defaultskewchar=1
\_hfuzz=0.1pt
\_vfuzz=0.1pt
\_overfullrule=5pt
\_linepenalty=10 % penalty between lines inside the paragraph
\_hyphenpenalty=50 % when a word is bro-ken
\_ezhyphenpenalty=50 % when the hyphenmark is used explicitly
\_binoppenalty=700 % between binary operators in math
\_relpenalty=500 % between relations in math
\_brokenpenalty=100 % after lines if they end by a broken word.
\_displaywidowpenalty=50 % before last line of paragraph if display math follows
\_predisplaypenalty=10000 % above display math
\_postdisplaypenalty=0 % below display math
\_delimiterfactor=901 % parameter for scaling delimiters
\_delimitershortfall=5pt
\_nulldelimiterspace=1.2pt
\_scriptspace=0.5pt % \texttt{\_setmathdimen}, \_setunimathdimen instead
\_maxdepth=4pt
\_splitmaxdepth=\_maxdimen
\_boxmaxdepth=\_maxdimen
\_parskip=0pt plus 1pt
\_abovedisplayskip=12pt plus 3pt minus 9pt
\_abovedisplayshortskip=0pt plus 3pt
\_belowdisplayskip=12pt plus 3pt minus 9pt
\_belowdisplayshortskip=7pt plus 3pt minus 4pt
```
Note that \topskip and \splittopskip are changed when first \typosize sets the main values (default font size and default \baselineskip).

2.7.2 Plain \TeX registers

Allocate registers that are used just like in plain \TeX. \smallskipamount, \medskipamount, \bigskipamount, \normalbaselineskip, \normallineskip, \normallineskiplimit, \jot, \interdisplaylinepenalty, \interfootnotelinepenalty.

Plain \TeX macros for setting parameters. \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 Op\TeX is living in Europe. If you set \enlang hyphenation patterns then \nonfrenchspacing is set.

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

\plaintexsetting(%
\emergencystretch=0pt
\clubpenalty=150
\widowpenalty=150
\pdfvorigin=1in
\pdfhorigin=1in
\hoffset=0pt
\voffset=0pt
\hsize=6.5in
\vsize=8.9in
\pagewidth=8.5 true in
\pageheight=11 true in
\nonfrenchspacing
} 
\public \plaintexsetting ;

2.7.4 OpTeX parameters

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

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

The \picdir tokens list can include a directory where image files (loaded by \inspic) are saved. Empty \picdir (default value) means that image files are in the current directory (or somewhere in the TeX system where 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 \picw) and \picheight. By default these parameters are set to zero: the native dimension of the image is used. If only \picwidth has a nonzero value, then this is the width of the image (height is calculated automatically in order to respect the aspect of the image). If only \picheight has a nonzero value then the height is given, the width is calculated. If both parameters are non-zero, the height and width are given and the aspect ratio of the image is (probably) broken. We recommend setting these parameters locally in the group where \inspic is used in order to not influence the dimensions of other images. But there exist many situations you need to put the same dimensions to more images, so you can set this parameter only once before more \inspic macros.

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

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

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

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

The \ttindent gives default indentation of verbatim lines printed by \begtt...\endtt pair or by \verbinput. The \ttshift gives the amount of shift of all verbatim lines to the right. Despite the \ttindent, it does not shift the line numbers, only the text.

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

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

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

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

The \everyitem tokens list is applied in vertical mode at the start of each item. The \everylist tokens list is applied after the group is opened by \begitems The \ilevel keeps the value of the current nesting level of the items list. The \listskipamount gives vertical skip above and below the items list if \ilevel=1.

The \tit macro includes \vglue\titskip above the title of the document.
The \begmulti and \endmulti pair creates more columns. The parameter \colsep declares the space between columns. If $n$ 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 \bibtexthook tokens list is used inside the group when \usebib command is processed after style file is loaded and before printing bib-entries. You can re-define a behavior of the style file here or you can modify the more declaration for printing (fonts, baselineskip, etc.) or you can define specific macros used in your .bib file.

The \biboptions is used in the iso690 bib-style for global options, see section 2.32.5.

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

\everycaptionf is used before printing caption in figures and \everycaptiont is used before printing caption in tables.

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

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

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

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(size) is used.
The \texttt{\textbackslash eqalign} macro can be configured by \texttt{\textbackslash eqlines} and \texttt{\textbackslash eqstyle} tokens lists. The default values are set in order this macro behaves like in Plain TeX. The \texttt{\textbackslash eqspace} is horizontal space put between equation systems if more columns in \texttt{\textbackslash eqalign} are used.

```
\newtoks \eqlines \eqlines ={\openup\jot}
\newtoks \eqstyle \eqstyle ={\strut\displaystyle}
\newdimen \eqspace \eqspace =20pt
\public \eqlines \eqstyle \eqspace ;
```

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

```
\newtoks \lmfil \lmfil ={\hfil}
\public \lmfil ;
```

The output routine uses token lists \texttt{\headline} and \texttt{\footline} in the same sense as plain TeX does. If they are non-empty then \texttt{\hfil} or \texttt{\hss} must be here because they are used inside \texttt{\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 \texttt{\fontdef\rmfixed\rm}, \texttt{\fontdef\itfixed\it}. Then use them in headline and footline:

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

```
\newtoks \headline \headline ={}
\newtoks \footline \footline ={\hss \rmfixed \folio \hss}
\public \headline \footline ;
```

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

```
\newdimen \headlinedist \headlinedist =14pt
\newdimen \footlinedist \footlinedist =24pt
\public \headlinedist \footlinedist ;
```

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

```
\newskip \pgbottomskip \pgbottomskip =0pt \relax
\public \pgbottomskip ;
```

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

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

51
This example sets current page with empty headline, but next pages have non-empty headlines.

\newtoks \nextpages
\public \nextpages ;

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

\texttt{\pgbackground={\Yellow \hrule height \Opt \depth \pdfpageheight width \pdfpagewidth}}

\newtoks \pgbackground \pgbackground ={} % for page background
\public \pgbackground ;

The parameters used in \texttt{\inoval} and \texttt{\incircle} macros can be re-set by \texttt{\ovalparams}, \texttt{\circleparams} tokens lists. The default values (documented in the user manual) are set in the macros.

\newtoks \ovalparams
\newtoks \circleparams
\ovalparams={\_roundness=2pt \_fcolor=\Yellow \_lcolor=\Red \_lwidth=.5bp
\_shadow=\N \_overlapmargins=\N \_hhkern=\Opt \_vvkern=\Opt }
\circleparams={\_ratio=1 \_fcolor=\Yellow \_lcolor=\Red \_lwidth=.5bp
\_shadow=\N \_overlapmargins=\N \_hhkern=3pt \_vvkern=3pt}
\newdimen \roundness \roundness=5mm % used in \clippingoval macro
\public \ovalparams \circleparams \roundness ;

Op\TeX\ defines “Standard Op\TeX\ markup language”\footnote{2.8 More Op\TeX\ macros} which lists selected commands from chapter 1 and gives their behavior when a converter from Op\TeX\ document to HTML or Markdown or \LaTeX\ is used. The structure-oriented commands are selected here, but the commands which declare typographical appearance (page layout, dimensions, selected font family) are omitted. More information for such a converter should be given in \texttt{\cnvinfo{⟨data⟩}}. Op\TeX\ simply ignores this but the converter can read its configuration from here. For example, a user can write:

\texttt{\cnvinfo {type=html, ⟨cnv-to-html-data⟩}}
\texttt{\cnvinfo {type=markdown, ⟨cnv-to-markdown-data⟩}}

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

\let\cnvinfo=\ignoreit

2.8 More Op\TeX\ macros

The second bundle of Op\TeX\ macros is here.

\texttt{\_codedecl \eoldef \{Op\TeX\ useful macros <2022-11-24}\} % preloaded in format

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

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

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

\footnote{Will be developed in 2021.}
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 \catcode table. 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 \[ filename-list \] loads files specified in comma separated \filename-list. 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:\[filename\] 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 [opt default] \[params\] \{replacement text\} defines the \macro with the optional parameter followed by normal parameters declared in \params. The optional parameter must be used as the first parameter in brackets [...]. If it isn’t used then \opt default is taken into account. The \replacement text can use \the\opt because optional parameter is saved to the \opt tokens register. Note the difference from \LaTeX concept where the optional parameter is in \#1. \OpTeX uses \#1 as the first normal parameter (if declared).

The \nospaceafter ignores the following optional space at expand processor level using the negative \romannumeral trick. The \nospacefuturelet 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 reads the text to the end of the current line and ignores it.

The `\scantoeol` macro ⟨text to end of line⟩ scans the ⟨text to end of line⟩ in verbatim mode and runs the `\macro`⟨⟨text to end of line⟩⟩. The `\macro` can be defined `\macro#1\{\scantextokens{#1}\}`. The new tokenization of the parameter is processed when the parameter is used, no when the parameter is scanned. This principle is used in definition of `\chap`, `\sec`, `\secc` and `_Xtoc` macros. It means that user can write `\sec text `\&` text` for example. Inline verbatim works in title sections.

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 \_replacestringsA\#1⟨⟨textA⟩⟩ and runs \_replacestringsA{foo-body}⟨⟨textA⟩⟩ before first ⟨textA⟩. It is saved to \_tmptoks and \_replacestringsB is run in a loop. It finishes processing or appends the next part to \_tmptoks separated by ⟨textB⟩ 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:

\def\replAB#1{\immediateassigned{\def\tmp{#1}\replstring\tmp{A}{B}}\the\_tmptoks}
\replAB {text A \and A} % expands to “text B \and B”
The `\catcode` primitive is redefined here. Why? There is very common cases like `\catcode`\langle\text{something}\rangle or `\catcode`\langle\text{number}\rangle but these characters ` or " can be set as active (typically by `\verbchar` macro). Nothing problematic happens if re-defined `\catcode` is used in this case.

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

The `\removespaces` \langle\text{with spaces}\rangle \{\text{\}} expands to \langle\text{without spaces}\rangle. The `\_\ea\ignorept`\langle\text{\dimen}\rangle expands to a decimal number \langle\text{\dimen}\rangle but without \text{pt} unit.

If you do `\let\foo=a` then it is not simple to return from \text{\foo} to the original character code of \text{a}. You can write `\text{a}` but you cannot write `\text{\foo}`. The macro `\cstochar`\langle\text{sequence}\rangle 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 `\expanded{`\text{\cstochar}\text{\foo}} if you want to extract the character code.

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

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

You can use expandable `\expr`\langle\text{\expression}\rangle for analogical purposes. It expands to the value of the \langle\text{\expression}\rangle at expand processor level with \_\decdigits digits after the decimal point. The \langle\text{\expression}\rangle can include \text{+-*/() \text{\and \text{decimal numbers}} in \text{common syntax}. The usage of prefixed versions \_\expr or \_\bp is more recommended because a user can re-define the control sequences \_\expr or \_\bp.

You can write `\setpos`\langle\text{\label}\rangle somewhere and the position of such `\setpos`\langle\text{\label}\rangle can be referenced by `\posx`\langle\text{\label}\rangle, `\posy`\langle\text{\label}\rangle and `\pospg`\langle\text{\label}\rangle. The first two macros expand to \text{x} and \text{y} position measured from left-bottom corner of the page (dimen values) and `\pospg`\langle\text{\label}\rangle expands to the \langle\text{\gpageno}\rangle, 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 \text{.ref} file and restored from it at the beginning of the \TeX job. If these values are not known then mentioned macros expand.
to 0sp, 0sp and 0. The following example implements \linefrom{⟨label⟩} and \lineto{⟨label⟩} macros. The line connecting these two points is drawn (after second \TeX run):

\def\linefrom[#1]{\setpos[#1:f]\drawlinefromto[#1]}\def\lineto[#1]{\setpos[#1:t]}\def\drawlinefromto[#1]{\ifnum\pospg[#1:f]>0\ifnum\pospg[#1:f]=\pospg[#1:t]\pdfliteral{q 0 0 m 1 0 0 RG % << red color}\expr{\bp\posx[#1:t]}-\bp\posx[#1:f]}\expr{\bp\posy[#1:t]}-\bp\posy[#1:f]}1 S Q\fi\fi}

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 .ref file in the format \_Xpos{⟨label⟩}{⟨x-pos⟩}{⟨y-pos⟩}. The \_Xpos macro defines \_pos{⟨label⟩} 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 \_pos, \posx and \posy macros are based on \padsavepos \pdflastxpos and \pdflastypos pdf\TeX primitives. The \pospg simply reads the data from the \_currpage macro.

\_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}{}}}{\_def\_posy[#1]{\_ea\_posii\_expanded{\_trycs{\_pos:#1}{}}}{\_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 ;

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

\_doc (ignored text) \\
\_cod (documentation)

The \_cod text too.

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

\_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.

\_def\_docgen #1 {\_ea\_docgenA\_input{#1.opm}}\_long\_def\_docgenA #1\_codedecl\_endcode #3\_doc {#1\_wlog(\_banner)\_skiptoeol}

\_public \docgen ;

2.9 Using key=value format in parameters

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

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

\myframe [margins=5pt,rule-width=2pt,frame-color=\Red,text-color=\Blue] {text1} or \myframe [frame-color=\Blue] {text2} % other parameters are default or simply \myframe {text3}. You can define \myframe as follows:

\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:
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\myframedefaults 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. The \kv\{⟨key⟩\} returns ??? if such key is not declared.

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

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

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

2.10 Plain \TeX macros

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

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

\addto \dospecials\{do\_}
The shortcuts \chardef\@one is not defined in 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.

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

Penalties macros: \break, \nobreak, \allowbreak, \filbreak, \goodbreak, \eject, \supereject, \dosupereject, \removelastskip, \smallbreak, \medbreak, \bigbreak.
Boxes. `\line`, `\leftline`, `\rightline`, `\centerline`, `\rlap`, `\llap`, `\underbar`.

Alignment. `\hidewidth`, `\ialign`, `\multispan`.

Tabbing macros are omitted because they are obsolete. Indentation and others. `\textindent`, `\item`, `\itemitem`, `\narrower`, `\raggedright`, `\ttraggedright`, `\leavevmode`.

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

\begin{verbatim}
\chardef\%=`\%
\_let\% = \_pcent % more natural, can be used in lua codes.
\chardef\&=`\&
\chardef\#=`\#
\chardef\$=`\$
\chardef\ss=`FF
\chardef\ae=`E6
\chardef\oe=`F7
\chardef\o=`F8
\chardef\AE=`C6
\chardef\OE=`D7
\chardef\O=`D8
\chardef\j=`1A % dotless letters
\chardef\aa=`E5
\chardef\AA=`C5
\chardef\S=`9F
\_def\l{\_errmessage{\_usedirectly ł}}
\_def\L{\_errmessage{\_usedirectly Ł}}
%\def\_{\_ifmmode \kern.06em \vbox{\hrule width.3em}\else _\fi} % obsolete
\_def\_\{\_hbox{_}}
\_def\dag{\_errmessage{\_usedirectly †}}
\_def\ddag{\_errmessage{\_usedirectly ‡}}
\_def\copyright{\_errmessage{\_usedirectly ©}}
%\_def\Orb{\_mathhexbox20D} % obsolete (part of Copyright)
%\_def\P{\_mathhexbox27B} % obsolete
\_def \_usedirectly #1{Load Unicoded font by \string\fontfam\space and use directly #1}
\_def \_mathhexbox #1#2#3{\_leavevmode \_hbox{$\_math \_mathchar"#1#2#3$}}
\_public \mathhexbox ;
\end{verbatim}

The \texttt{\_unichars} macro is run in \texttt{\initunifonts}, Unicodes are used instead old plain TeX settings.

\begin{verbatim}
\_def\_unichars{% Plain TeX character sequences with different codes in Unicode:
\chardef\ss=`ß
\_chardef\ae=`æ \_chardef\AE=`Æ
\_chardef\oe=`œ \_chardef\OE=`Œ
\_chardef\o=`ø \_chardef\O=`Ø
\_chardef\aa=`å \_chardef\AA=`Å
\_chardef\l=`ł \_chardef\L=`Ł
\_chardef\i=`ı \_chardef\j=`ȷ
\_chardef\S=`§ \_chardef\P=`¶
\_chardef\dag`†
\_chardef\ddag`‡
\_chardef\copyright`©
}%
\_def\_oalign #1{\_leavevmode\_vtop{\_baselineskip=\zo \_lineskip=.25ex
\_ialign{##\_crcr#1\_crcr}}}
\_def\_oalignA {\_lineskiplimit=\zo \_oalign}
\_def\_oalign {\_lineskiplimit=\_maxdimen \_oalign} % chars over each other
\_def\_shiftx #1{\_dimen0=#1\_kern\_ea\_ignorept \_the\_fontdimen1\_font
\_dimen0 } % kern by #1 times the current slant
\_def\_c #1{\_oalign{\_\char\_maxdimen\relax\_crcr\_hidewidth\shiftx(-1ex)\_hidewidth}}
\_public \oalign \ooalign \d \b \c \dots ;
\end{verbatim}

The accent commands like \texttt{\_v}, \texttt{\_\}, \texttt{\_H}, etc. are not defined. Use the accented characters directly – it is the best solution. But you can use the macro \texttt{\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.
The last part of plain \TeX macros: \texttt{\textbackslash magnification, \textbackslash bye}. Note that math macros are defined in the \texttt{math-macros.opm} file (section 2.15).

\begin{verbatim}
\def \magnification {\afterassignment \magA \count255 }
\def \magA {\mag=\count255 \truedimen\hsize \truedimen\vsize
\dimen\footins=8truein}

% only for backward compatibility, but \margins macro is preferred.
\public \magnification ;

\def \showhyphens #1{\setbox0=\vbox{\parfillskip=0pt \hsize=\maxdimen \tenrm
\pretolerance=-1 \tolerance=-1 \hbadness=0 \showboxdepth=0 \#1}}

\def \bye {\par \vfill \supereject \byehook \end}
\public \showhyphens \bye ;
\end{verbatim}

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

\begin{verbatim}
\lefthyphenmin=2 \righthyphenmin=3 \% disallow x- or -xx breaks
\input hyphen % en(USenglish) patterns from TeX82
\end{verbatim}

\section{11 Preloaded fonts for text mode}

The format in Lua\TeX can download only non-Unicode fonts. Latin Modern EC is loaded here. These fonts are totally unusable in Lua\TeX 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 \texttt{\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 \texttt{\everyjob}. But why to load a set of fonts at the beginning of every job when it is highly likely that the user will load something completely different. Our decision is: there is a basic 8bit font set in the format (for testing purposes only) and the user should load a Unicode font family at beginning of the document.

The fonts selectors \texttt{\tenrm, \tenbf, \tenit, \tenbi, \tentt} are declared as \texttt{\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 succesfully generated without any pre-loaded fonts. It doesn’t matter if each document processed by Op\TeX declares Unicode fonts. You can create such fonts-less format anyway if you set \texttt{\fontspreload} to \texttt{\relax} before \texttt{\input optex.ini}, i.e.: \texttt{\textbackslash luatex -ini \textbackslash let\textbackslash fontspreload=\relax \input optex.ini}'

\begin{verbatim}
\codedecl \tenrm {Latin Modern fonts (EC) preloaded <2022-02-12>} \% preloaded in format
\ifx\fontspreload\relax
\set\tenrm=\nullfont \set\tenbf=\nullfont \set\tenit=\nullfont \set\tenbi=\nullfont
\else
\% Only few text fonts are preloaded:
\% allow missing fonts during format generation
\ suppressfontnotfounderror=1 \font\tenrm=ec-lmr10 \% roman text
\ font\tenbf=ec-lmbx10 \% boldface extended
\ font\tenit=ec-lmri10 \% text italic
\ font\tenbi=ec-lmbxi10 \% bold italic
\ font\tentt=ec-lmtt10 \% typewriter
\ suppressfontnotfounderror=0
\fi
\tenrm
\public \tenrm \tenbf \tenit \tenbi \tentt ;
\end{verbatim}
2.12 Using \font primitive directly

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

\font \langle font switch \rangle = \langle font file name \rangle \langle size spec \rangle

% for example:
\font \tipa = tipa10 at12pt % the font tipa10 at 10pt is loaded
% usage:
\{\tipa TEXT} % the TEXT is printed in the loaded font.

The \langle size spec \rangle can be empty or at\langle dimen \rangle or scaled\langle scale factor \rangle. The \langle font file name \rangle must be terminated by space or surrounded in the braces.

OpTeX starts with \font primitive which is able to read only tfm files. i.e. the \langle font file name \rangle.tf\m (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 \langle font switch \rangle = \{\langle font file name \rangle: \langle font features \rangle\} \langle size spec \rangle
% or
\font \langle font switch \rangle = \{\langle font name \rangle: \langle font features \rangle\} \langle size spec \rangle

where \langle font file name \rangle is name of the OpenType font file without extension (extensions .otf or .ttf are assumed). The braces in the syntax are optional, use them when the \langle font file name \rangle or \langle font name \rangle 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 luaotfload package. More information is available in the luaotfload latex.pdf file.

You can use \ufont macro which runs \initunifonts followed by \font primitive. And \fontfam does (among other things) \initunifonts too. You need not to specify \initunifonts if \fontfam or \ufont is used.

It seems that you must decide about final size of the font before it is loaded by the \font primitive. It is not exactly true; OpTeX offers powerful possibility to resize the font already loaded on demand. See the example at the end of next subsection.

2.12.1 The \setfontsize macro

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

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

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

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

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

The \setfontsize command acts like font modifier. It means that it saves information about fonts but does not change the font actually until variant selector or \resizethefont is used.
The following example demonstrates the `mag` format of \setfontsize parameter. It is only a curious example probably not used in practical typography.

\def\smaller{\setfontsize{mag.9}\resizethefont}
Text \smaller text \smaller text \smaller text.

The \resizethefont works with arbitrary current font, for example with the font loaded directly by \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 \crimson 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 \sfont with the same syntax like \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 \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.

\begin{verbatim}
\fontlet ⟨new font switch⟩ = ⟨given font switch⟩ ⟨size spec⟩
\end{verbatim}

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 \_regoptsizes command. The command \setfontsize sets the internal requirements for optical size if the parameter is in the format \at⟨dimen⟩ or \mag⟨factor⟩. Then the command \resizethefont or \fontlet or variant selectors try to choose the font suitable for the required optical size. For example

\begin{verbatim}
\fontfam[lm]
The text is printed in font [lmroman10-regular] at 10 pt.
\setfontsize{at13pt}\rm
Now, the text is printed in [lmroman12-regular] at 13 pt.
\end{verbatim}

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 2.13.4.

\setfontsize {Font resizing macros <2022-11-08>} % 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 \pdftex and \luatex. Luaotfload uses Lua functions from \luatexbase namespace, we provide our own replacements. \initunifonts sets itself to relax because we don’t want to do this work twice. \ufont is a shortcut of \initunifonts \font.

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

\newdimen \optsize \optsize=10pt
\newdimen \defaultoptsize \defaultoptsize=10pt
\newdimen \lastmagsize
\def \setoptsize #1{\edef \sizenspec\{\setoptsizeA\}}
\edef \sizenspec\{\setoptsizeA\}
\pdfstringdef\setoptsizeA{\imextchar a\{\setoptsizeC\}}{\setoptsizeB}{\setoptsizeC}{\setoptsizeB}}
\def \setoptsizeA at\{dimen\} \setoptsize\{\defaultoptsize\} \setoptsize\{\lastmagsize\} \setoptsize\{\optsize\} \setoptsize\{\scalewidth\} \setoptsize\{\scaleheight\}
\def \setoptsizeB scaled\{scale\} \setoptsize\{\defaultoptsize\} \setoptsize\{\lastmagsize\} \setoptsize\{\optsize\} \setoptsize\{\scalewidth\} \setoptsize\{\scaleheight\}
\def \setoptsizeC mag\{mag\} \setoptsize\{\defaultoptsize\} \setoptsize\{\lastmagsize\} \setoptsize\{\optsize\} \setoptsize\{\scalewidth\} \setoptsize\{\scaleheight\}
\edef \sizenspec\{\at\{dimen\}\optsize\}
\public \setfontsize \defaultoptsize ;
The \texttt{\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 ⟨dimen⟩} from \texttt{\fontname} output. The letters \texttt{at} have category 12.

\begin{Verbatim}
\edef\stringat{\string a\string t}\end{Verbatim}

\texttt{\fontlet ⟨font switch A⟩ ⟨font switch B⟩ ⟨size spec⟩} does

\begin{Verbatim}
\font ⟨font switch A⟩ = \{⟨font file name⟩\} ⟨size spec⟩
\end{Verbatim}

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

Optical sizes data for preloaded 8bit Latin Modern fonts:
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, they do appropriate change in the font context but do not select the font.

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

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

The family context is the current family 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.

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:

⟨family selector⟩ ⟨font modifiers⟩ ⟨variant selector⟩

For example:
\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.  

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:  
\rm default size \setfontsize{at14pt}\rm here is 14pt size \it italic is in 14pt size too \bf bold too.  

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

2.13.3 Math Fonts  
Most font families are connected with a preferred Unicode-math font. This Unicode-math is activated when the font family is loaded. If you don’t prefer this and you are satisfied with 8bit math CM+AMS fonts preloaded in the OpTEX format then you can use command \noloadmath before you load a first font family.  
If you want to use your specially selected Unicode-math font then use \loadmath {{⟨font file⟩}} or \loadmath ⟨⟨font name⟩⟩ before first \fontfam is used.  

2.13.4 Declaring font commands  
Font commands can be font switches, variant selectors, font modifiers, family selectors and defined font macros doing something with fonts.  
• Font switches can be declared by \font primitive (see section 2.12) 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 \langle font-switch \rangle by the \fontdef command.  
\fontdef\langle font-switch \rangle \{\langle family selector \rangle \langle font modifiers \rangle \langle variant selector \rangle\}\}  
where \langle family selector \rangle and \langle font modifiers \rangle are optional and \langle variant selector \rangle is mandatory.  
The resulting \langle font-switch \rangle declared by \fontdef is “fixed font switch” independent of the font context. More exactly, it is a fixed font switch when it is used. But it can depend on the current font modifiers and font family and given font modifiers when it is declared.
The \fontdef does the following steps. It pushes the current font context to a stack, it does modifications of the font context by given \(\text{(family selector)}\) and/or \(\text{(font modifiers)}\) and it finds the real font by \(\text{(variant selector)}\). This font is not selected but it is assigned to the declared \(\text{(font switch)}\) (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\langle new variant selector \rangle \{\langle family selector \rangle \langle font modifiers \rangle \langle variant selector \rangle\}

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

\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 Op\TeX internal macros \ttfont (verbatim texts) and \urlfont (printing URL’s).

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

```latex
\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 \tt stay family independent. This is a special feature only for \tt selector. New definitions of \ttfont and \urlfont are done too. It is recommended to use \setff{-liga;-tlig} to suppress the ligatures in typewriter fonts.

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

2.13.8 Font commands defined by \def

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

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

```latex
\def\titlefont{\setfontsize{at14pt}\bf \let\it\bi}
```

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

\OpTeX defines similar internal commands \titfont, \chapfont, \secfont and \seccfont, see section 2.26. The commands \typsize and \boldify are used in these macros. They set the math fonts to given size too and they are defined in section 2.17.

2.13.9 Modifying font features

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

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

You can declare a new variant selector too:

```latex
\fontfam[LinLibertine]
\famvardef\fraclig{\setff{+frac}\currvar}
```

Compare 1/2 or 1/10 \fraclig to 1/2 or 1/10.

If the used font does not support the given font feature then the font is reloaded without warning nor error, silently. The font feature is not activated.

The onum font feature (old-style digits) is connected to \caps macro for Caps+SmallCaps variant in Op\TeX font family files. So you need not create a new modifier, just use \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):
The \setfontsize{\langle size spec \rangle} \% sets the font size
\setff{\langle font feature \rangle} \% adds the font feature
\setletterspace{\langle number \rangle} \% sets letter spacing
\setwordspace{\langle scaling \rangle} \% modifies word spacing

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

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

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

You can use \setff with other font features provided by LuaTeX and luaotfload package (see documentation of luaotfload package for more information):
\setff{embolden=1.5} \rm \% font is bolder because outline has nonzero width
\setff{slant=0.2} \rm \% font is slanted by a linear transformation
\setff{extend=1.2} \rm \% font is extended by a linear transformation.
\setff{color=yes} \rm \% if the font includes colored characters, use colors
\setff{upper} \rm \% to uppercase \langle lower=lowecase \rangle conversion at font level
\setff{fallback=name} \rm \% use fonts from a list given by name if missing chars

Use font transformations embolden, slant, extend and \setletterspace, \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 \setff{-liga} to disable ligatures. And setting a positive letter spacing probably needs to scale interword spacing too.

All mentioned font modifiers (except for \setfontsize) work only with Unicode fonts loaded by \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:

```
f-heros.opm

\fdecl [Heros] \Heros {TeX Gyre Heros fonts based on Helvetica}
\{\caps \cond \rm \bf \it \bi \FiraMath\}
\{[texgyreheros-regular]\}
\_def \fontnamegen{[texgyreheros-\condV-\currV]::[\capsV]\_fontfeatures}\n
\wlog{\_detokenize{\
Modifiers:"\J \caps ...... caps & small caps~\} \ cond ...... condensed variants~\J \)}
\_initfontfamily \% new font family must be initialized
\_ifmathloading
\_loadmath {[FiraMath-Regular]}\n\_addUmathfont \_xits {[XITSMath-Regular]}{} {[XITSMath-Bold]}{}{}
\_addto\_frak\_\_fam\_xits\_\_addto\_cal\_\_fam\_xits\_\_public \frak \cal ;
\% \bf, \bi from FiraMath:
\_let \bsansvariables=\_bfvariables
\_let \bsansGreek=\_bfGreek
\_let \bsansgreek=\_bfgreek
```
If you want to write such a font family file, you need to keep the following rules.

1. Use the \_famdecl command first. It has the following syntax:

```
\_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 ⟨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.

2. You can use \_wlog{\_detokenize{...} to write additional information into a log file.

3. 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.

4. Declare font modifiers using \moddef if they are present. The \resetmod must be declared in each font family.

5. 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.

6. Optionally, declare new variants by the \famvardef macro.

7. Run \_initfontfamily to start the family (it is mandatory).

8. If math font should be loaded, use \_loadmath{⟨math font⟩}.

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}
```

and define the \_fontnamegen in the last parameter of the \_famdecl by:

```
\_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-{⟨family⟩}.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 \{[texgyreheros-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 \_\{word\}_V are preferred. They expand typically to their content. The macro \_\fsetV \{word\}={\{content\}} (terminated by a space) is equivalent to \def\_\{word\}V{\{content\}} and you can use it in font modifiers. You can use the \_\fsetV macro in more general form:
\_\fsetV \{word-a\}={\{value-a\}}, \{word-b\}={\{value-b\}} ... etc. terminated by a space

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

Example 3
If both font modifiers \caps, \cond were applied in Heros family, then \def\_\capsV{+smcp;\_ffonum;} and \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 \{word\}={\{value-a\}}, {\{value-b\}} ... {\{value-n\}}: {\{what\}}. It can be used inside \moddef as simple IF statement: the \{what\} is processed only if \{word\} has \{value-a\} or \{value-b\} ... or \{value-n\}. 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, \setletterspace macros can work.

The \_moddef macro
has the syntax \moddef\{modifier\}>{\{what to do\}}. 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 \{Familyselector\} is defined by the \_famdecl macro as:
\protected\def\{Familyselector\} \{
\_def\_currfamily {\{Familyselector\}}%
\_def\_fontnamegen {...} % this is copied from 7-th parameter of \_famdecl
\resetmod
\{run all family-dependent font modifiers used before Familyselector without warnings\}

The \_initfontfamily
must be run after modifier's declaration. It runs the \{Familyselector\} and it runs \_\rm, so the first font from the new family is loaded and it is ready to use it.

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

The name of \{Familyselector\} 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.

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

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

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

In general:
\_regoptsizes \langle internal-template \rangle \langle general-output-template \rangle \langle resizing-data \rangle 

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

If \_optname gets an argument which is not registered \langle internal-template \rangle then it expands to \_failedoptname which typically ends with an error message about missing font. You can redefine \_failedoptname macro to some existing font if you find it useful.

We are using a special macro \_LMregfont in f-lmfonts.opm. It sets the file names to lowercase and enables us to use shortcuts instead of real \langle resizing-data \rangle. There are shortcuts \_regoptFS, \_regoptT, etc. here. The collection of \langle internal-templates \rangle are declared, each of them covers a collection of real file names.

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

The \_optname macro can be used even if no optical sizes are provided by a font family. Suppose that font file names are much more chaotic (because artists are very creative people), so you need to declare more systematic \langle internal-templates \rangle and do an alias from each \langle internal-template \rangle to \langle real-font-name \rangle. For example, you can do it as follows:
\def\fontalias #1 #2 {\_regoptsizes #1 ?#2 {}} 

% alias name   real font name
74
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 \texttt{\textbackslash osize} font modifier:

\begin{verbatim}
\fontfam \[Poltawski\] \Poltawski {Antykwa Poltawskiego, Polish traditional font family} \\
\texttt{\_def \_fontnamegen \{[antpolt\_liV\_condV-\_currV]\_capsV\_fontfeatures\}} \\
\_wlog\{\_detokenize{\% Modifiers:^^J \\
\light ..... light weight, \bf,\bi=semibold^^J \\
\noexpd .... no expanded, no condensed, designed for 10pt size (default)^^J \\
\expd ..... expanded, designed for 6pt size^^J \\
\cond ..... semi condensed, designed for 12pt size^^J \\
\ccond ..... condensed, designed for 17pt size^^J \\
\osize ..... auto-sitches between \ccond \cond \noexpd \expd \eexpd by size^^J \\
\caps ...... caps & small caps^^J \\
}} \\
\_moddef \_resetmod \{ \_fsetV li={},\cond={},\caps={} \_fvars regular bold italic bolditalic \} \\
\_moddef \light \{_fsetV li=lt \} \\
\_moddef \noexpd \{ \_fsetV cond={} \} \\
\_moddef \expd \{ \_fsetV cond=expd \} \\
\_moddef \cond \{ \_fsetV cond=semicond \} \\
\_moddef \ccond \{ \_fsetV cond=cond \} \\
\_moddef \caps \{ \_fsetV caps=\_smcp;\_ffonum; \} \\
\_moddef \nocaps \{ \_fsetV caps={} \} \\
\_moddef \osize \{ \_def \_fontnamegen\{[antpolt\_liV\_optname{x}-\_currV]\\_capsV\_fontfeatures\} \%
\_regoptsizes x ? expd <7 semiexpd <9 {} <11.1 semicond <15 cond <>\}
\_initfontfamily % new font family must be initialized
\end{verbatim}

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 \texttt{f-⟨famname⟩.opm} and \TeX can see it in your filesystem then you can type \texttt{\fontfam[⟨famname⟩]} and the file is read, so the information about the font family is loaded. The name \texttt{⟨famname⟩} must be lowercase and without spaces in the file name \texttt{f-⟨famname⟩.opm}. On the other hand, the \texttt{\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 \texttt{\fontfam \[LM Fonts\]} is equivalent to \texttt{\fontfam \[LMfonts\]} and both commands load the file \texttt{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 \texttt{\fontfam[?]} is used and it is not included in the font catalog when \texttt{\fontfam[catalog]} is printed. The list of families taken in the catalog and listed on the terminal is declared in two files: \texttt{fams-ini.opm} and \texttt{fams-local.opm}. The second file is optional. Users can create it and write to it the information about user-defined families using the same syntax as in existed file \texttt{fams-ini.opm}.

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

The \texttt{fams-ini.opm} and \texttt{fams-local.opm} files can use the macros \texttt{\_faminfo}, \texttt{\_famalias} and \texttt{\_famtext}. See the example from \texttt{fams-ini.tex}:
The \_faminfo command has the syntax:
\_faminfo [{\FamilyName}] {\{comments\}} {\{file-name\}}
\{\mod-plus-vars\}

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

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

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

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

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 \ttt 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 \langle font switch \rangle in the format \_ten\langle famv \rangle and loads the font associated to the \langle font switch \rangle. The loading is done by:

a) \letfont \langle font switch \rangle = \savedswitch \_sizespec
b) \font \langle font switch \rangle = \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 \langle font-switch \rangle.

If a font is loaded by macros \fontsel or \resizethefont then the \_fontloaded \langle font switch \rangle is called immediately after it. If the font is loaded first then its \skewchar is equal to −1. We run \_newfontloaded \langle font switch \rangle 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 OpTeX trick 0058.

The \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.

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

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\{data\} \rangle runs \initunifonts, then checks if \langle Famselector \rangle
is defined. If it is true, then closes the file by \endinput. Else it defines \fontnamegen{\famselector} and saves it to the internal \f:\currfamily:\main.fam command. The macro \initfontfamily needs it. The \currfamily is set to the \famselector because the following \moddef commands need to be in the right font family context. The \currfamily is set to the \famselector by the \famdepend too, because a \famselector must set the right font family context. The font family context is given by the current \currfamily value and by the current meaning of the \fontnamegen macro. The \mathfaminfo is saved for usage in the catalog.

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

\currV expands to a template declared by \fvars depending on the (variant name). Useable only of standard four variants. Next variants can be declared by the \famvardef macro.

\fsetV {key}=\{value\},...,{key}=\{value\} expands to \def{\{key\}}V\{\{value\}\} in the loop.

\onlyif {key}=\{value-a\},...,{key}=\{value-z\} 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=\ignoreit to ignore it 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:\currentfamily} {\{\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.

\fontdef (font-switch) \{\langle data\rangle\}\ does:

\begingroup \langle data\rangle \ea \endgroup \ea \let \ea (font-switch) \the\font

It means that font modifiers used in \langle data\rangle are applied in the group and the resulting selected font (current at the end of the group) is set to the \langle font-switch\rangle. We want to declare \langle font-switch\rangle in its real name directly by \font primitive in order to save this name for reporting later (in overfull messages, for example). This is the reason why \_xfamv and \_ttfamv are re-defined locally here. They have precedence when \fontsel constructs the \langle font switch\rangle name.

The \famvardef \xxx \{\langle data\rangle\} does, roughly speaking:

\def \xxx \{\langle data\rangle\}\ea the\font \def \_currvavari\xxx\}

but the macro \xxx is declared as family-dependent. It is analogically as in \moddef. The \xxx is defined as \_famdepend \xxx\{f:\langle currfamily\rangle:xxx\} and \_f:\langle currfamily\rangle: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 `f-⟨fontfamily⟩.opm` exists read it and finish.
- Try to load user defined `fams-local.opm`.
- If the `{fontfamily}` is declared in `fams-local.opm` or `fams-ini.opm` read relevant file and finish.
- Print the list of declared families.

The `fams-local.opm` is read by the `\tryloadfamslocal` macro. It sets itself to `\relax` because we need not load this file twice. The `\listfamnames` macro prints registered font families to the terminal and to the log file.

When the `fams-ini.opm` or `fams-local.opm` files are read then we need to save only a mapping from family names or alias names to the font family file names. All other information is ignored in this case. But if these files are read by the `\listfamnames` macro or when printing a catalog then more information is used and printed.

`\famtext` does nothing or prints the text on the terminal.

`\faminfo` [{Family Name}] {{comments}} {{file-name}} {{mod-plus-vars}} does `\famf:` `{familyname} {{file-name}}` (only if `{file-name}` differs from `f-{familyname}`) or prints information on the terminal. The `{mod-plus-vars}` data are used when printing the font catalog.

`\famalias` [{Family Alias}] does `\famf:` `{familyalias} {{file-name}}` where `{file-name}` is stored from the previous `\faminfo` command. Or prints information on the terminal.

`\famfrom` declares type foundry or owner or designer of the font family. It can be used in `fams-ini.opm` or `fams-local.opm` and it is printed in the font catalog.
When the \fontfam[catalog] is used then the file fonts-catalog.opm is read. The macro \faminfo is redefined here in order to print catalog samples of all declared modifiers/variant pairs. The user can declare different samples and different behavior of the catalog, see the end of catalog listing for more information. The default parameters \catalogsample, \catalogmathsample, \catalogonly and \catalogexclude of the catalog are declared here.

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

- \defaultfontfeatures – used for each font,
- \ffadded – features added by \setff,
- \ffcolor – features added by \setfontcolor (this is obsolette)
- \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 \setletterspace is based on the special font features provided by luaotfload package. The \setwordspace recalculates the \fontdimen2,3,4 of the font using the \setwsp macro which is used by the \fontselA macro. It activates a dummy font feature +Ws too in order the font is reloaded by the \font primitive (with independent \fontdimen registers). \setfontcolor is kept here only for backward compatibility but not recommended. Use real color switches and the \transparency instead.
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 \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 OptiTeX says: set three sizes by the macro \setmathsizes \[ ⟨text-size⟩/⟨script-size⟩/⟨scriptscript-size⟩ \] and then load all math fonts in given sizes by \normalmath or \boldmath macros. For example
\setmathsizes[12/8.4/6] \normalmath ... math typesetting at 12 pt is ready.

We have two math macros \normalmath for the normal shape of all math symbols and \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 \mit, \cal, \bbchar, \frak and \script are defined here. The \rm, \bf, \it, \bi and \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 unimath-codes.omp is loaded, see the section 2.16.2.

The optical sizes of Computer Modern fonts, AMS, and other fonts are declared here.
\_loadmathfamily \langle \text{number} \rangle \langle \text{font} \rangle \text{loads one math family, i.e. the triple of fonts in the text size, script size and script-script size. The} \langle \text{font-id} \rangle \text{is used in the } \_regtfm \text{ parameter or the real TFM name. The family is saved as } \_\text{mfam}. \_\setmathfamily \langle \text{number} \rangle \langle \text{font-switch} \rangle \text{loads one math family like } \_\text{loadmathfamily} \text{ does it. But the second parameter is a } \langle \text{font-switch} \rangle \text{ declared previously by the } \_\text{font} \text{ primitive. The } \langle \text{number} \rangle \text{ is saved by } \_\text{loadmathfamily}, \_\text{setmathfamily} \text{ to the } \_\text{mfam}. \_\text{mfactor} \text{ sets scaling factor for given math fonts family related to text font size. It does the setting } \_\text{ptunit}=(\langle \text{factor} \rangle) \_\text{ptunit} \text{ where the } \langle \text{factor} \rangle \text{ is defined by } \_\text{sdef}\langle \text{mfactor}:\langle \text{family} \rangle \rangle \langle \text{factor} \rangle \text{. For example, you can set } \_\text{sdef}\langle \text{mfactor}:1\rangle\{0.95\} \text{ if you found that this scaling of math family } 1 \text{ gives better visual compatibility with used text fonts. If not declared then scaling factor is } 1. \_\text{ptmunit} \_\text{ptunit} ; \_\text{setmathparam} \langle \text{luatex-param} \rangle \{\langle \text{factor} \rangle\} \text{ sets } \langle \text{luatex-param} \rangle \text{ (like } \_\text{mathspaceafterscript} \text{) to values dependent on } \_\text{ptunit} \text{ of textfont, scriptfont, scriptscriptfont. The } \langle \text{factor} \rangle \text{ is scaling factor of mentioned } \_\text{ptunit}. \_\text{ptunit} \_\text{ptunit} \_\text{ptunit} \_\text{ptunit} ;
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 10pt typesetting). The \_skewchar of some math families are set here too.

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

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

\_mathsbon macro activates the rewriting rule \langle word \rangle \_langle one-letter \rangle (where \langle word \rangle is a sequence of letters) because such control sequences are inaccessible: preprocessor rewrites it.

The \_mathsboff deactivates it. You can ask by \_ifmathsb if this feature is activated or deactivated. By default, it is activated in the \everyjob, see section 2.1. Note, that the \everyjob is processed after the first line of the document is read, so the \_mathsbon is activated from the second line of the document.

\_catcode\_ = 8 \let\sb = _
\_catcode\_ = 13 \let_ = \sb
\_catcode\_ = 11
\_private \sb ;
\_newif\_ifmathsb \_mathsbfalse
\_def\_mathsbfalse %
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.

\begin{verbatim}
\_mathcode`\^^@="2201 % \cdot
\_mathcode`\^^A="3223 % \downarrow
\_mathcode`\^^B="010B % \alpha
\_mathcode`\^^C="010C % \beta
\_mathcode`\^^D="225E % \land
\_mathcode`\^^E="023A % \lnot
\_mathcode`\^^F="3232 % \in
\_mathcode`\^^G="0119 % \pi
\_mathcode`\^^H="0115 % \lambda
\_mathcode`\^^I="010D % \gamma
\_mathcode`\^^J="010E % \delta
\_mathcode`\^^K="3222 % \uparrow
\_mathcode`\^^L="2206 % \pm
\_mathcode`\^^M="2208 % \oplus
\_mathcode`\^^N="0231 % \infty
\_mathcode`\^^O="0140 % \partial
\_mathcode`\^^P="321A % \subset
\_mathcode`\^^Q="321B % \supset
\_mathcode`\^^R="225C % \cap
\_mathcode`\^^S="225B % \cup
\_mathcode`\^^T="0238 % \forall
\_mathcode`\^^U="0239 % \exists
\_mathcode`\^^V="220A % \otimes
\_mathcode`\^^W="3224 % \leftrightarrow
\_mathcode`\^^X="3220 % \leftarrow
\_mathcode`\^^Y="3221 % \rightarrow
\_mathcode`\^^Z="8000 % \ne
\_mathcode`\^^[="2205 % \diamond
\_mathcode`\^^\="3214 % \le
\_mathcode`\^^\="3215 % \ge
\_mathcode`\^^\="3211 % \equiv
\_mathcode`\^^\="225F % \lor
\_mathcode`\^^\="8000 % \space
\_mathcode`\^^\="5021
\_mathcode`\^^\="8000 % \prime
\_mathcode`\^^\="4028
\_mathcode`\^^\="5029
\_mathcode`\^^\="2203 % \ast
\_mathcode`\^^\="202B
\_mathcode`\^^\="613B
\_mathcode`\^^\="2200
\_mathcode`\^^\="013A
\_mathcode`\^^\="603B
\_mathcode`\^^\="313C
\_mathcode`\^^\="303D
\_mathcode`\^^\="313E
\_mathcode`\^^\="503F
\_mathcode`\^^\="605B
\_mathcode`\^^\="026E % \backslash
\_mathcode`\^^\="505D
\_mathcode`\^^\="8000 % math-active subscript
\end{verbatim}
All control sequences declared by \mathchardef are supposed (by default) only for public usage. It means that they are declared without _ prefix. If such sequences are used in internal Op\TeX macro then their internal prefixed form is declared using \_private macro. These encoding dependent declarations will be set to different values when Unicode-math font is loaded. The declared sequences for math symbols are not hyperlinked in this documentation.

The math functions like log, sin, cos are declared in the same way as in plain\TeX, but they are \protected in Op\TeX.
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 ^\texttt{\sb} is an alternative for _ was defined at line 27 of the file \texttt{math-macros.opm}.

\texttt{\let\sp=} ^ \texttt{\sb=} , defined at beginning of this file

Active \texttt{\prime} character is defined here.

\texttt{\big, \Big, \bigg, \Bigg, \bigl, \bigm, \bigr, \Bigr, \biggl, \biggm, \biggr, \Biggl, \Biggm, \Biggr} are based on the \texttt{\scalebig} macro because we need the dependency on the various sizes of the fonts.

Math relations defined by the \texttt{\jointrel} plain \TeX macro:
\ldots, \cdots, \vdots, \ddots from plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

\ldots \cdots \vdots \ddots \ldots inspired by plain \TeX

Math accents (encoding dependent declarations).
Macros based on \delimiter, \*witdelims and \radical primitives.

\protected\def\lmoustache{\delimiter"437A340 } % top from (, bottom from )
\protected\def\rmoustache{\delimiter"537B341 } % top from ), bottom from ()
\protected\def\lgroup{\delimiter"462833A } % extensible ( with sharper tips
\protected\def\rgroup{\delimiter"562933B } % extensible ) with sharper tips
\protected\def\arrowvert{\delimiter"26A33C } % arrow without arrowheads
\protected\def\Arrowvert{\delimiter"26B33D } % double arrow without arrowheads
\protected\def\bracevert{\delimiter"77C33E } % the vertical bar that extends braces
\protected\def\Vert{\delimiter"26B30D } \let\|=\Vert
\protected\def\vert{\delimiter"26A30C }
\protected\def\uparrow{\delimiter"3222378 }
\protected\def\downarrow{\delimiter"3223379 }
\protected\def\updownarrow{\delimiter"326C33F }
\protected\def\Uparrow{\delimiter"322A37E }
\protected\def\Downarrow{\delimiter"322B37F }
\protected\def\Updownarrow{\delimiter"326D377 }
\protected\def\backslash{\delimiter"26E30F } % for double coset G\backslash H
\protected\def\langle{\delimiter"426830A }
\protected\def\rangle{\delimiter"526930B }
\protected\def\lbrace{\delimiter"4266308 } \let\lbrace=\lbrace
\protected\def\rbrace{\delimiter"5267309 } \let\rbrace=\rbrace
\protected\def\{{\ifmmode\lbrace\else\char`{\fi}
\protected\def\}{\ifmmode\rbrace\else\char`}\fi}
\protected\def\rceil{\delimiter"5265307 }
\protected\def\lceil{\delimiter"4264306 }
\protected\def\rfloor{\delimiter"5263305 }
\protected\def\lfloor{\delimiter"4262304 }
\protected\def\choose{\atopwithdelims()}\protected\def\brack{\atopwithdelims\[\]}
\protected\def\brace{\atopwithdelims\lbrace\rbrace}
\protected\def\sqrt{\radical"270370 } \public \sqrt ;
\mathpalette, \vphantom, \hphantom, \phantom, \mathstrut, and \smash macros from plain \TeX.
\texttt{\cong, \notin, \rightleftharpoons, \buildrel, \doteq, \bmod} and \texttt{\pmod} macros from plain \TeX.

\texttt{\texttt{math-macros.opm}}

\begin{verbatim}
573 \protected\def\_cong{\_mathrel{\_mathpalette\_overeq\_sim}} % congruence sign
574 \def\overeq#1#2{\lower.05em\vbox{\lineskiplimit\maxdimen\lineskip=-.05em
575 \ialign{$\_math#1\hfil##\hfil$\crcr#2\crcr\crcr}}}
576 \protected\def\_notin{\_mathrel{\_mathpalette\_cancel\_in}}
577 \def\cancel#1#2{\_math\ooalign{$\hfil#1\mkern1mu/\hfil$\crcr$#1#2$}}
578 \protected\def\_rightleftharpoons{\_mathrel{\_mathpalette\_rlhp{}}}
579 \def\rlhp#1{\vcenter{\_math\hbox{\_ooalign{\_raise.2em
580 \hbox{$#1\rightharpoonup$}\_crcr$#1\leftharpoondown$}}}}
581 \protected\def\_buildrel#1\over#2{\_mathrel{\_mathop{\_kern\zo #2}
582 \limits^{#1}}}
583 \protected\def\_doteq{\_buildrel\textstyle.\over=}
584 \_private \in \sim ;
585 \_public \cong \notin \rightleftharpoons \buildrel \doteq ;
586 \protected\def\_bmod{\_nonscript\mskip-\medmuskip\mkern5mu
587 \mathbin{\_\rm mod}\penalty900\mkern5mu\nonscript\mskip-\medmuskip}
588 \protected\def\_pmod#1{\_allowbreak\mkern18mu({\_\rm mod}\_thinsk\_thinsk#1)}
589 \_public \bmod \pmod ;
\end{verbatim}

\texttt{\texttt{\textbackslash matrix and \textbackslash pmatrix}\ behave as in Plain \TeX, if it is used in the \texttt{\textbackslash displaystyle}. On the other hand, it is printed in smaller size (by appropriate amount) in \texttt{\textbackslash textstyle = \textbackslash scriptstyle} and \texttt{\textbackslash scriptscriptstyle}. This feature is new in \Op\TeX.}

\texttt{\texttt{\textbackslash pmatrix} behave as in Plain \TeX. You can simply re-define \texttt{\textbackslash bordermatrix} with other delimiters using the common \texttt{\_bordermatrixwithdelims} macro.}

\texttt{\texttt{\textbackslash cases and \textbackslash \textbackslash bordermatrix macros are almost identical as in plain \TeX. You can simply re-define \texttt{\textbackslash bordermatrix} with other delimiters using the common \texttt{\_bordermatrixwithdelims} macro.}}
The \align macro behaves like in Plain TeX by default. It creates the \center in the math mode. The content is two column \halign with right-aligned left column and left-aligned right column. The table items are in \displaystyle and the \baselineskip is advanced by \jot (3pt in plain TeX). It follows from the default settings of \eqlines and \eqstyle parameters.

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

There are more possible columns than two (used in classical Plain TeX): rl, cr, lcrlcrlcrlc etc. where r and l columns are without spaces and c column (if used) has space /2 at its both sides.

\begin{verbatim}
\_long\_def\_eqalign#1{\_null\_thinsk\_vcenter{\_the\_eqlines\_math
\_align(##\_the\_eqstyle{##}##\_hfil
\&\_hskip.5\_eqspace\_hfil##\_hskip.5\_eqspace\_hfil
\&\_crcr#1\_crcr}}}\_thinsk
\end{verbatim}

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

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

\end{verbatim}

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

\begin{verbatim}
\_def\_amsafam4 \_def\_amsbfam5
\_mathchardef \boxdot "2\_amsafam 00
\_mathchardef \boxplus "2\_amsafam 01
\end{verbatim}
The \_not macro is re-defined to be smarter than in plain \TeX. The macro follows this rule:

\[
\begin{align*}
\text{\_not<} & \text{ becomes } \_\text{nless} \\
\text{\_not>} & \text{ becomes } \_\text{ngtr} \\
\text{if } \_\text{notXXX} \text{ is defined, } \text{\_not\XXX} & \text{ becomes } \_\text{notXXX; } \\
\text{if } \_\text{nXXX} \text{ is defined, } \text{\_not\XXX} & \text{ becomes } \_\text{nXXX; } \\
otherwise, \text{\_not\XXX} & \text{ is done in the usual way.}
\end{align*}
\]

\text{\_mathchardef \_notchar "3236}
\text{\protected\_def \_not#1{\_ifx #1<\_nless \_else}
\_ifx #1>\_ngtr \_else}
\_edef\_tmpn{\csstring#1}\\
\_ifcsname _not\_tmpn\_endcsname \_csname _not\_tmpn\_endcsname
\_else \_ifcsname _n\_tmpn\_endcsname \_csname _n\_tmpn\_endcsname
\_else \_mathrel{\_mathord{\_notchar}\_mathord{#1}}\\
\_fi \_fi \_fi \_fi}
\_private
\_nleq \_ngeq \_nless \_ngtr \_nprec \_nsucc \_nleqslant \_ngeqslant \_npreceq
\_nsuccneq \_nleq \_ngeq \_nsim \_ncong \_nsubseteqq \_nsupseteqq \_nsubseteq
\_nsupseteq \_unparallel \_uplus \_usubscript \_usubscripteq
\_usubsetq \_usubscript \_usubscripteq \_usubscripteq
\_vdash \_nvdash \_trianglelefteq \_trianglerighteq \_triangleleft \_triangleright
\_nexists \_nleftarrow \_nrightarrow \_nLeftarrow \_nRightarrow
\_nleftrightarrow \_nexists \_nexists
\_public \not ;
\]

\text{\_mathstyles{\{\mathlist\}}} behaves like \{\mathlist\}, but you can use the following commands in the \{\mathlist\}:

- \text{\_currstyle} which expands to \displaystyle, \textstyle, \scriptstyle or \scriptscriptstyle depending on the current math style when \mathstyles was opened.
- \text{\_dobystyle\{D\}\{T\}\{S\}\{SS\}} is expandable macro. It expands to \{D\}, \{T\}, \{S\} or \{SS\} depending on the current math style when \mathstyles was opened.
- The value of the \text{\_stylenum} is 0, 1, 2 or 3 depending on the current math style when \mathstyles was opened.

Example of usage of \mathstyles: \text{\_def\_mathframes#1\{\_mathstyles\{\frame\{$\_\text{currstyle#1}\$}\}\}}.

\text{\_mathchardef \_boxtimes "2\_amsafam 02}
\text{\_mathchardef \_square "0\_amsafam 03}
\text{\_mathchardef \_blacksquare "0\_amsafam 04}
\text{\_mathchardef \_centerdot "2\_amsafam 05}
\text{\_mathchardef \_lozenge "0\_amsafam 06}
\text{\_mathchardef \_blacklozenge "0\_amsafam 07}
\text{\_mathchardef \_circlearrowright "3\_amsafam 08}
\text{\_mathchardef \_circlearrowleft "3\_amsafam 09}
\text{\_mathchardef \_rightleftharpoons "3\_amsafam 0A}
\text{\_mathchardef \_leftrightharpoons "3\_amsafam 0B}
\text{\_mathchardef \_boxminus "2\_amsafam 0C}
\]

\text{\_protected\_def \_not\_#1{\_ifx #1<\_nless \_else}
\_ifx #1>\_ngtr \_else}
\_edef\_tmpn{\csstring#1}\\
\_ifcsname _not\_tmpn\_endcsname \_csname _not\_tmpn\_endcsname
\_else \_ifcsname _n\_tmpn\_endcsname \_csname _n\_tmpn\_endcsname
\_else \_mathrel{\_mathord{\_notchar}\_mathord{#1}}\\
\_fi \_fi \_fi \_fi}
\_private
\_nleq \_ngeq \_nless \_ngtr \_nprec \_nsucc \_nleqslant \_ngeqslant \_npreceq
\_nsuccneq \_nleq \_ngeq \_nsim \_ncong \_nsubseteqq \_nsupseteqq \_nsubseteq
\_nsupseteq \_unparallel \_uplus \_usubscript \_usubscripteq
\_usubsetq \_usubscript \_usubscripteq \_usubscripteq
\_vdash \_nvdash \_trianglelefteq \_trianglerighteq \_triangleleft \_triangleright
\_nexists \_nleftarrow \_nrightarrow \_nLeftarrow \_nRightarrow
\_nleftrightarrow \_nexists \_nexists
\_public \not ;
\]

\text{\_mathstyles{\{\mathlist\}}} behaves like \{\mathlist\}, but you can use the following commands in the \{\mathlist\}:

- \text{\_currstyle} which expands to \displaystyle, \textstyle, \scriptstyle or \scriptscriptstyle depending on the current math style when \mathstyles was opened.
- \text{\_dobystyle\{D\}\{T\}\{S\}\{SS\}} is expandable macro. It expands to \{D\}, \{T\}, \{S\} or \{SS\} depending on the current math style when \mathstyles was opened.
- The value of the \text{\_stylenum} is 0, 1, 2 or 3 depending on the current math style when \mathstyles was opened.

Example of usage of \mathstyles: \text{\_def\mathframe#1\{\_mathstyles\{\frame\{$\_\text{currstyle#1}\$}\}\}}.

\text{\_newcount\_stylenum}
\text{\_def\_mathstyles#1{{\_mathchoice{\_{\_stylenum0}#1}{\_stylenum1#1}{\_stylenum2#1}{\_stylenum3#1}}}}
\text{\_def\_dobystyle#1#2#3#4{\_ifcase\_stylenum#1\_or#2\_or#3\_or#4\_fi}}
\text{\_def\_currstyle{\_dobystyle{\_displaystyle}{\_textstyle}{\_scriptstyle}{\_scriptscriptstyle}}}
\text{\_public \_dobystyle \_currstyle \_stylenum ;}

The \text{\_cramped} macro sets the cramped variant of the current style. Note that \text{\_currstyle} initializes non-cramped variants. The example \text{\_mathframe} above should be:
\text{\_def\mathframe#1\{\_mathstyles\{\frame\{$\_\text{currstyle#1} \_\text{cramped}$\}$\}\}}.

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

93
The \mathbox{⟨text⟩} macro is copied from OPmac trick 078. It behaves like \hbox{⟨text⟩} but the ⟨text⟩ is scaled to a smaller size if it is used in scriptstyle or scriptscript style. The \textmff and \scriptmff are redefined in order to respect optical sizes. If we are in script style then the math mode starts in text style, but optical size is given to script style. The \mathbox in non-Unicode math respects optical sizes using different principle.

2.16 Unicode-math fonts

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

\loadboldmath {⟨bold-font⟩} to {⟨normal-font⟩} loads bold variant only if ⟨normal-font⟩ was 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 ⟨normal-font⟩ 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 {\xitsmath-regular}
\addUmathfont \famname {\xitsmath-bold} \to {\xitsmath-regular}

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

\fontfam[1m] \% does \loadmath{\latinmodern-math}
\addUmathfont \famname {{XITSMath-Regular}} {{XITSMath-Bold}} {{XITSMath-Regular}}
\addUmathfont \famname {{XITSMath-Regular}} {{XITSMath-Bold}}{\famname}

declares latinmodern-math as main math font (its bold variant is “faked bold”). The additional math font family \xits 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 re-declare characters for printing from additional font by \mathchars \famname ⟨list of sequences⟩. For example:

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

sets the characters \stareq, \triangleq, \veeeq, \wedgeq from the \xits additional font. The ⟨list of sequences⟩ 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{\famname} if you want to print whole math alphabet from an additional math font. For example \addto\cal{\famname} declares all \cal characters from the \xits font loaded by \addUmathfont.
The \mathcodes macro provides comfortable settings of math codes of math objects. Its syntax is \mathcodes \langle family \rangle \{ \langle list-of-pairs \rangle \}. Each pair in the \langle list-of-pairs \rangle is \langle class-number \rangle \langle character \rangle (separated by optional space) or \langle class-number \rangle \{ \langle list-of-characters \rangle \}. The \langle list-of-characters \rangle 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 \mathcodes macro declares mathcode of given characters internally by \mathcode `\langle character \rangle = \langle class-number \rangle \langle family \rangle `\langle character \rangle.

The \mathcodes 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

\loadmath \{ \langle Unicode-math font \rangle \} \% preloaded in format

\loadmath \{ \langle Unicode-math font \rangle \} loads the given font. It does:

- define \unimathfont as \langle Unicode-math font \rangle,
- redefine \normalmath and \boldmath macros to their Unicode counterparts,
- load the \unimathfont by \normalmath,
- print information about the loaded font on the terminal,
- redefine all encoding dependent setting by \input unimath-codes.opm,
- protect new loading by setting \ifmathloading to false.

\noloadmath disallows Unicode-math loading by \mathloadingfalse.
\doloadmath allows Unicode-math loading by \mathloadingtrue.

\loadboldmath \{ \langle bold-font \rangle \} \to \{ \langle normal-font \rangle \} defines \unimathboldfont as \langle bold-font \rangle only if \unimathfont is defined as \langle normal-font \rangle. It is used when \boldmath macro is run. When no \unimathboldfont is defined then the \boldmath macro use “fake bold” generated by embolden LuaTeX font feature.
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 \setmathdimens. 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.

\begin{verbatim}
\def\normalunimath{%
  \setmathfamily 0 \tenrm % font for non-math objects in math mode
  \loadmathfamily 1 {\unimathfont}{} % Base font
  \loadmathfamily 4 rsfs % script
  \setunimathdimens
}\def\boldunimath{%
  \setmathfamily 0 \textbf % font for non-math objects in math mode
  \ifx\unimathboldfont \undefined
    \loadumathfamily 1 {\unimathfont}{\textbf} % Base faked bold
  \else
    \loadumathfamily 1 {\unimathboldfont}{} % Base real bold font
  \fi
  \loadmathfamily 4 rsfs % script
  \setunimathdimens
}\def\setunimathdimens{% PlainTeX sets these dimens for 10pt size only:
  \delimitershortfall=0.5\fontdimen6\textfont1
  \nulldelimiterspace=0.12\fontdimen6\textfont1
  \setmathparam\Umathspaceafterscript\scriptspacefactor
  \setbox0=\hbox{$\scriptstyle 0\atop0$}\
  \Umathfractiondelsize\displaystyle=\dimexpr(\ht0-\Umathaxis\displaystyle)*2\relax
  \setbox0=\voidbox
}\end{verbatim}

If you try the example above about \loadboldmath\{\textit{math-bold}\} \to \{\textit{math-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:

\begin{verbatim}
\loadmath{\textit{math-regular}}
\def\boldmath{%
  \loadmathfamily 1 {\textit{math-bold}}{} % Base font
  \loadmathfamily 4 rsfs % script
  \loadmathfamily 5 {\textit{math-regular}}{}
  \def\{|\Udelimiter 0 5 "02016 }% % norm delimiter from family 5
  \setunimathdimens
}\end{verbatim}

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

\texttt{\_mparams} can insert additional font features depending on the current \texttt{\_mfam}. The \texttt{\_mfactor \{\textit{family}\}\{\textit{space}\}} sets scaling factor, see section 2.14 for more information.

The \texttt{\_textmff}, \texttt{\_scriptmff} and \texttt{\_sscriptmff} are font features for text, script and scriptscript sizes respectively. They are locally re-defined in \texttt{\mathbox} macro.
Unimath macros include a variety of math alphabets, which are encoded by different parts of the Unicode table. We need auxiliary macros for setting mathcodes by selected math alphabet.

### \_umathrangefirst

\_umathrangefirst \{from\} \{to\} \{class\} \{family\} \{first\} sets the \texttt{Umathcodes} of the characters in the interval \{from\} \{to\} to \{first\}, \{first\}+1, \{first\}+2 etc., but \texttt{umathcharholes} are skipped. The \texttt{umathrangefirst} \{from\} \{to\} \{first\} is the same as \texttt{umathrange} \{\texttt{alpha}\} \{\texttt{omega}\} \{first\}.

### \_umathrangefirstGREEK

\_umathrangefirstGREEK \{first\} is the same as \texttt{umathrange} \{\texttt{Alpha}\} \{\texttt{Omega}\} \{first\}.

### \_greekdef (control sequences) \texttt{\_relax}

\texttt{\_relax} defines each control sequence as a normal character with codes from \_umathnumB, \_umathnumB+1, \_umathnumB+2 etc. It is used for redefining the control sequences for math Greek \texttt{\alpha}, \texttt{\beta}, \texttt{\gamma} etc.

### \_umathnum

\_umathnumA \_umathnumB

\_umathnum starts from 43 because numbers 1–42 are reserved for direct usage without \texttt{newfam} and \texttt{boldmath} macros. Note that allocationos using \texttt{newfam} starts from 43 because numbers 1–42 are reserved for direct usage without \texttt{newfam}. We use \texttt{\_headto} here because we want to read the main family 1 as last one (for definitive setting of math parameters).
\mathchars \{\text{fam}\} \{\langle \text{list of sequences}\rangle\} \text{saves \langle \text{fam}\rangle to} \ \text{\_mafam} \text{and runs for each sequence from the} \ \langle \text{list of sequences}\rangle \text{the relevant code settings. The \_directlua \text{chunk prints the 8-digits hexadecimal code of the sequence followed by the sequence itself. The digits are scanned by} \ \_mathcharsB \text{and new} \ \text{\Umathcode} \text{is declared. If there exist}: \langle \text{sequence}\rangle \text{(i.e. it is delimiter) then} \ \text{\_Udelcode} \text{is redeclared too. Finally, in case of} \ \text{\_intop} \text{like operators the} \ \text{\_intop} \ \langle \text{sequence}\rangle \text{is redeclared by} \ \text{\Umathchardef}. \text{Note that the used primitives have the syntax:}

\text{\Umathchardef} \langle \text{sequence}\rangle = \langle \text{math class}\rangle \langle \text{math family}\rangle \langle \text{slot number}\rangle \\
\text{\Umathcode} \langle \text{code}\rangle = \langle \text{math class}\rangle \langle \text{math family}\rangle \langle \text{slot number}\rangle \\
\text{\_Udelcode} \langle \text{code}\rangle = \langle \text{math family}\rangle \langle \text{slot number}\rangle 

\mathcodes \langle \text{fam}\} \{\langle \text{list of pairs}\rangle\} \text{sets mathcodes of given characters with explicit} \ \langle \text{class}\rangle\text{es. Each pair can be} \ \langle \text{class}\rangle \{\langle \text{list of chars}\rangle\} \text{and} \ \langle \text{list of chars}\rangle \text{can include} \ \text{\_range} \langle \text{from}\rangle \langle \text{to}\rangle. \text{This is reason why we apply} \text{\_expanded} \text{to the} \ \langle \text{list of chars}\rangle \text{before reading it by} \ \text{\_foreach}: \text{the} \ \text{\_range} \text{is expandable and expands to the relevant list of characters.}

\text{\_def}_{\ \text{mathcode}} \langle \text{code}\rangle \langle \text{math class}\rangle \langle \text{math family}\rangle \langle \text{slot number}\rangle

\text{\_cond} \langle \text{class}\rangle \langle \text{list of chars}\rangle

2.16.2 Macros and codes set when \loadmath is processed firstly

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

\text{\_codedecl} \text{\_ncharrmA} \langle \text{Uni math codes <2022-11-20}> \rangle \% preloaded on demand by \loadmath

The control sequences for \text{\_alpha}, \text{\_beta} etc are redefined here. The \text{\_alpha} \text{expands to the character with Unicode “03B1”, this is a normal character} \alpha. \text{You can type it directly in your editor if you know how to do this.}
The math alphabets are declared here using the \umathrange{\langle range\rangle}{\langle class\rangle}{\langle family\rangle}{\langle starting-code\rangle} macro.

\chardef\ncharrmA=`A \chardef\ncharrma=`a
\chardef\ncharbfA=`1D400 \chardef\ncharbfa=`1D41A
\chardef\ncharitA=`1D434 \chardef\ncharita=`1D44E
\chardef\ncharbiA=`1D468 \chardef\ncharbia=`1D482
\chardef\ncharclA=`1D49C \chardef\ncharcla=`1D4B6
\chardef\ncharbcA=`1D4D0 \chardef\ncharbca=`1D4EA
\chardef\ncharfrA=`1D504 \chardef\ncharfra=`1D51E
\chardef\ncharbrA=`1D56C \chardef\ncharbra=`1D586
\chardef\ncharbbA=`1D538 \chardef\ncharbba=`1D552
\chardef\ncharsnA=`1D5A0 \chardef\ncharsna=`1D5B6
\chardef\ncharbsA=`1D5D4 \chardef\ncharbsa=`1D5EE
\chardef\ncharsiA=`1D608 \chardef\ncharsia=`1D622
\chardef\charsxA=`1D63C \chardef\charsxa=`1D656
\chardef\charttA=`1D670 \chardef\chartta=`1D68A

\protected\def\rmvariables {\umathrange{A-Z}\ncharrmA}\umathrange{a}\ncharrma}
\protected\def\bfvariables {\umathrange{A-Z}\ncharbfA}\umathrange{a}\ncharbfa}
\protected\def\itvariables {\umathrange{A-Z}\ncharitA}\umathrange{a}\ncharita}
\protected\def\bivariables {\umathrange{A-Z}\ncharbiA}\umathrange{a}\ncharbia}
\protected\def\calvariables {\umathrange{A-Z}\ncharclA}\umathrange{a}\ncharcla}
\protected\def\bcalvariables {\umathrange{A-Z}\ncharbcA}\umathrange{a}\ncharbca}
\protected\def\frakvariables {\umathrange{A-Z}\ncharfrA}\umathrange{a}\ncharfra}
\protected\def\bfrakvariables {\umathrange{A-Z}\ncharbrA}\umathrange{a}\ncharbra}
\protected\def\bbvariables {\umathrange{A-Z}\ncharbbA}\umathrange{a}\ncharbba}
\protected\def\sansvariables {\umathrange{A-Z}\ncharsnA}\umathrange{a}\ncharsna}
\protected\def\bsansvariables {\umathrange{A-Z}\ncharbsA}\umathrange{a}\ncharbsa}
\protected\def\isansvariables {\umathrange{A-Z}\ncharsiA}\umathrange{a}\ncharsia}
\protected\def\biosansvariables {\umathrange{A-Z}\ncharsxA}\umathrange{a}\ncharsxa}
\protected\def\ttvariables {\umathrange{A-Z}\ncharttA}\umathrange{a}\nchartta}

\protected\def\itgreek {\umathrangegreek\ncharrma} \protected\def\rmgreek {\umathrangegreek\ncharrmA}
\protected\def\bfgreek {\umathrangegreek\ncharbfa} \protected\def\bigreek {\umathrangegreek\ncharbfa}
\protected\def\bsansgreek {\umathrangegreek\ncharsna} \protected\def\bisansgreek {\umathrangegreek\ncharsna}

\protected\def\itGreek {\umathrangeGREEK\ncharrma} \protected\def\rmGreek {\umathrangeGREEK\ncharrmA}
\protected\def\bfGreek {\umathrangeGREEK\ncharbfa} \protected\def\bigreek {\umathrangeGREEK\ncharbfa}
\protected\def\bsansGreek {\umathrangeGREEK\ncharsna} \protected\def\bisansGreek {\umathrangeGREEK\ncharsna}

\_setnabla is used in order to \nabla behaves like uppercase Greek letter, similar like \Delta. It depends on \bf, \it etc. selectors. If you want to deactivate this behavior, use \def\setnabla\#1 {}.

\_def \setnabla {\Umathcode"2207 = 7 1} \_def \setnablarm {\setnabla\"02207 } \_def \setnablaf {\setnabla\"1D6C1 } \_def \setnablat {\setnabla\"1D6FA } \_def \setnabla1 {\setnabla\"1D735 } \_def \setnablabs {\setnabla1\"1D76F } \_def \setnablabs1 {\setnabla1\"1D7A9 }

Digits are configured like math alphabets.
The math alphabets \texttt{\cal}, \texttt{\bbchar}, \texttt{\frak}, \texttt{\script} are re-defined here. The \texttt{\_marm}, \texttt{\_mabf}, \texttt{\_mait}, \texttt{\_mabi}, \texttt{\_matt} used in \texttt{\rm}, \texttt{\bf}, \texttt{\it}, \texttt{\bi} are re-defined too. You can redefine them again if you need different behavior (for example you don’t want to use sans serif bold in math). What to do:

\begin{verbatim}
\protected\def\mbabf {\_inmath\{\_bfvariables\_bfgreek\_bfGreek\_bfdigits\}}
\protected\def\mabi {\_inmath\{\_bivariables\_bigreek\_bfGreek\_bfdigits\}}
\end{verbatim}

\verb|\_protected\def\inmath#1{\_relax\_ifmmode#1\_fi}| % to keep off \loop processing in text mode

% You can redefine these macros to follow your wishes.
% For example, you need upright lowercase greek letters, you don’t need
% \texttt{\bf} and \texttt{\bi} behave as sans serif in math, ...

\begin{verbatim}
\protected\def\marm {\_inmath\{\_rmvariables \_rmdigits\}}
\protected\def\mait {\_inmath\{\_itvariables \_itgreek \_rmGreek\}}
\end{verbatim}

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

\begin{verbatim}
\begingroup \% input mathclass.opm (which is a copy of MathClass.txt):
\long\def\p#1;#2 {\_ifx\^#2\_else
\end{verbatim}
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:

```latex
\begin{verbatim}
\def\UnicodeMathSymbol #1#2#3#4{\chardef#2=#1 % control sequence is only pointer to Unicode character
  \ifnum#1=\Umathcodenum#1 \Umathcode#1=0 \fi % it isn’t set by mathclass.opm
  \ifx#3\mathopen \setdelimiter#2\fi
  \ifx#3\mathclose \setdelimiter#2\fi
  \ifx#3\mathfence \setdelimiter#2\fi
  \ifx#3\mathaccent \protected\def#2{\Umathaccent fixed 7 \char#1}
\end{verbatim}
```

We have to read this information and set given control sequences to the relevant Unicode characters using `\chardef`. We don’t use `\mathchardef` for them because the want to manage the matcodes (class, family, slot) only at single place: at Unicode characters. Control sequences declared here are only pointers to these Unicode characters. Exceptions are delimiters and math accents.

If the character is delimiter then we declare `⟨sequence⟩` to be the `\chardef` equivalent of the character and `⟨sequence⟩=⟨character⟩` because `\left`, `\right` doesn’t like `\chardef` equivalent of the character although it has its own delcode.

```latex
\begin{verbatim}
\global\Udelcode`<=1 `027E8 % these characters have different meaning
\global\Udelcode`>=1 `027E9 % as normal and as delimiter
\mit % default math alphabets setting
\def\Umathcode `~ = 2 \texttt{2212}
\def\setdelimiter#1\fi
\def\setmathaccent\protected\def#2{\Umathaccent fixed 7 \char#1}\fi
\end{verbatim}
```

Many special characters must be declared with care...

```latex
\begin{verbatim}
\ifnum#1=\Umathcodenum#1 \Umathcode#1=0 \fi % it isn’t set by mathclass.opm
\end{verbatim}
```

We don’t use `\mathchardef` for them because the want to manage the matcodes (class, family, slot) only at single place: at Unicode characters. Control sequences declared here are only pointers to these Unicode characters. Exceptions are delimiters and math accents.

If the character is delimiter then we declare `⟨sequence⟩` to be the `\chardef` equivalent of the character and `⟨sequence⟩=⟨character⟩` because `\left`, `\right` doesn’t like `\chardef` equivalent of the character although it has its own delcode.

```latex
\begin{verbatim}
\ifnum#1=\Umathcodenum#1 \Umathcode#1=0 \fi % it isn’t set by mathclass.opm
\end{verbatim}
```
We save the \mathcode of $\int$ to $\intop$ using \mathchardef and declare Unicode character $\int$ as math-active and define it as $\intop\nolimits$. We define $\intop$ as $\inttop$ for users (integral with normal limits). We do this with other int-like operators listed below too.

\begin{verbatim}
\protected\def \overbrace #1{\mathop {\mathaccent 7 1 "023DE{#1}}\limits}
\protected\def \underbrace #1{\mathop {\mathaccent bottom 7 1 "023DF{#1}}\limits}
\protected\def \overparen #1{\mathop {\mathaccent 7 1 "023DC{#1}}\limits}
\protected\def \underparen #1{\mathop {\mathaccent bottom 7 1 "023DD{#1}}\limits}
\protected\def \overbracket #1{\mathop {\mathaccent 7 1 "023B4{#1}}\limits}
\protected\def \underbracket #1{\mathop {\mathaccent bottom 7 1 "023B5{#1}}\limits}
\end{verbatim}

\begin{verbatim}
\def \nabla {\Huge \nabla} % \nabla behaves as uppercase Greek letter, see \setnabla
\end{verbatim}

\begin{verbatim}
\mathchardef \ldotp = "612E
\let \| = \Vert
\mathcode `/ = 0 1 `/ % mathclass says that / is Bin, Plain TeX says that it is Ord.
\end{verbatim}

\begin{verbatim}
\protected\def \vdots {\relax \ifnum \mathstyle>3 \unicodevdots \else \vdots \fi}
\protected\def \ddots {\relax \ifnum \mathstyle>3 \unicodeddots \else \ddots \fi}
\protected\def \adots {\relax \ifnum \mathstyle>3 \unicodeadots \else \adots \fi}
\end{verbatim}
% Unicode superscripts (²) and subscripts as simple macros with \mathcode"8000
\begingroup
\def\tmp#1#2{\_global\mathcode#1="8000 \_lccode`~=#1 \_lowercase\_gdef~}{#2}
\fornum 0..1 \do \_tmp{{207#1}}{{^#1}}
\_tmp{{B2}}{{^2}}\_tmp{{B3}}{{^3}}
\fornum 4..9 \do \_tmp{{207#1}}{{^#1}}
\fornum 0..9 \do \_tmp{{208#1}}{{_#1}}
\egroup

Aliases are declared here. They are names not mentioned in the unimath-table.opm file but commonly used in \TeX.

\let \setminus=\smallsetminus
\let \diamond=\smwhtdiamond
\let \colon=\mathcolon
\let \bullet=\smblkcircle
\let \circ=\vysmwhtcircle
\let \bigcirc=\mdlgwhtcircle
\let \to=\rightarrow
\let \le=\leq
\let \ge=\geq
\let \neq=\ne
\protected\def \triangle {\mathord{\bigtriangleup}}
\let \emptyset=\varnothing
\let \hbar=\hslash
\let \land=\wedge
\let \lor=\vee
\let \owns=\ni
\let \gets=\leftarrow
\let \mathring=\ocirc
\let \lnot=\neg
\let \longdivisionsign=\longdivision
\let \backepsilon=\upbackepsilon
\let \eth=\matheth
\let \dbkarow=\dbkarrow
\let \drbkarow=\drbkarrow
\let \hksearow=\hksearrow
\let \hkswarow=\hkswarrow
\let \square=\mdlgwhtsquare
\let \blacksquare=\mdlgblksquare
\let \upalpha=\mupalpha
\let \upbeta=\mupbeta
\let \upgamma=\mupgamma
\let \updelta=\mupdelta
\let \upepsilon=\mupvarepsilon
\let \upvarepsilon=\mupvarepsilon
\let \upzeta=\mupzeta
\let \upeta=\mupeta
\let \uptheta=\muptheta
\let \upiota=\mupiota
\let \upkappa=\mupkappa
\let \uplambda=\muplambda
\let \upmu=\mupmu
\let \upnu=\mupnu
\let \upxi=\mupxi
\let \upomicron=\mupomicron
\let \uppi=\muppi
\let \uprho=\muprho
\let \upvarrho=\mupvarrho
\let \upsilon=\mupsigma
\let \upvartheta=\mupvartheta
\let \upchi=\mupchi
\let \uppsi=\muppsi
\let \upomega=\mupomega
\let \upvartheta=\mupvartheta
\let \upalpha=\mupalpha
\let \upbeta=\mupbeta
\let \upgamma=\mupgamma
\let \updelta=\mupdelta
\let \upepsilon=\mupvarepsilon
\let \upvarepsilon=\mupvarepsilon
\let \upzeta=\mupzeta
\let \upeta=\mupeta
\let \uptheta=\muptheta
\let \upiota=\mupiota
\let \upkappa=\mupkappa
\let \uplambda=\muplambda
\let \upmu=\mupmu
\let \upnu=\mupnu
\let \upxi=\mupxi
\let \upomicron=\mupomicron
\let \uppi=\muppi
\let \uprho=\muprho
\let \upvarrho=\mupvarrho
\let \upsilon=\mupsigma
\let \upvartheta=\mupvartheta

103
The \texttt{\not} macro is redefined here. If the \texttt{\not\langle char\rangle} is defined (by \texttt{\negationof}) then this macro is used. Else centered / is printed over the \langle char\rangle.

\begin{verbatim}
\protected\def\not#1{\trycs{not\csstring#1}{\mathrel{\mathstyles{\setbox0=\hbox{$\currstyle#1$}\hbox to \wd0{\hss$\currstyle/$\hss}\kern-\wd0 \box0}}}}
\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\cong\ncong\negationof\leftarrow\nleftarrow\negationof\rightarrow\nrightarrow\negationof\leftrightarrow\nleftrightarrow\negationof\Leftarrow\nLeftarrow\negationof\Leftrightarrow\nLeftrightarrow\negationof\Rightarrow\nRightarrow\negationof\exists\nexists\negationof\ni\nni\negationof\parallel\nparallel\negationof\nequiv\nequiv\negationof\asymp\nasymp\negationof\lesssim\nlesssim\negationof\ngtrsim\ngtrsim\negationof\lessgtr\nlessgtr\negationof\gtrless\ngtrless\negationof\prec\nprec\negationof\succ\nsucc\negationof\subset\ns\subset\negationof\supset\ns\supset\negationof\subseteq\ns\subseteq\negationof\supseteq\ns\supseteq\negationof\vdash\nvdash\negationof\Vdash\nVdash\negationof\vartriangleleft\nvartriangleleft\negationof\vartriangleright\nvartriangleright\negationof\trianglelefteq\ntrianglelefteq\negationof\trianglerighteq\ntrianglerighteq\negationof\vinfty\nvinfty
\end{verbatim}

Newly declared public control sequences are used in internal macros by \TeX. We need to get new meanings for these control sequences in the private namespace.
2.16.3 More Unicode-math examples

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

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


2.16.4 Printing all Unicode math slots in used math font

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

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

\input print-unimath.opm

print-unimath.opm

\_undefined (Printing Unicode-math table \string<2020-06-08>)
\_undefined \_opwarning{No Unicode math font loaded, printing ignored}
\_endinput
\_def\UnicodeMathSymbol#1#2#3#4{\_ifnum#1>"10000 \_endinput \else \_printmathsymbol{#1}{#2}{#3}{#4}\_fi}
\_def\UnicodeMathSymbolA#1#2#3#4{\_ifnum#1>"10000 \_printmathsymbol{#1}{#2}{#3}{#4}\_fi}
\_def\_printmathsymbol#1#2#3#4{\_hbox\_hbox to2em{$#2{}$\_hss}\_hbox to3em{\small\_printop#3\_hss}{\_tt\_string#2\_trycs{_eq:\_string#2}{}}}
\_def\_eq#1#2{\_sdef{_eq:\_string#2}{\_eq#1\_string#2}}
\_eq\_diamond\smwhtdiamond\_eq\_bullet\smblkcircle\_eq\_circ\vysmwhtcircle\_eq\_bigcirc\mdlgwhtcircle\_eq\_to\rightarrow\_eq\_le\leq\_eq\_ge\geq\_eq\_ne\ne\_eq\_emptyset\varnothing\_eq\_hbar\hslash\_eq\_land\wedge\_eq\_lor\vee\_eq\_owns\ni\_eq\_gets\leftarrow\_eq\_mathring\ocirc\_eq\_inot\neg\_eq\_backepsilon\upbackepsilon\_eq\_eth\matheth\_eq\_dbkarow\_eq\_drbkarow\_eq\_hksearrow\_eq\_hkswarrow\_eq\_hkswarow\_eq\_hkswarrow
\_tracinglostchars=0
\_fontdef\small\_setfontsize{at5pt}\_rm
\_def\_printop{\_def\mathom{Op}}
\_def\mathom\_alpha\_def\mathom\_ord\_def\mathom\_bin\_def\mathom\_rel\_def\mathom\_re{Rel}}
\_def\mathom\_open\_def\mathom\_close\_def\mathom\_punct\_def\mathom\_fence\_fence}
\_def\mathom\_acc\_def\mathom\_accentvide\_def\mathom\_accent\_def\mathom\_over\_def\mathom\_under}
\_typosize[7.5/9]\_normalmath\_everymath{f}
\_codes U+00000 \_dots U+10000
\_begmulti 3
\_endmulti
\_medskip\_goodbreak
\_codes U+10001 \_dots U+1EEF1 \_let\UnicodeMathSymbol=\UnicodeMathSymbolA
\_begmulti 4
\_endmulti
105
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)/\baselineskip\} sets given parameters. It sets text font size by the \texttt{\setfontsize} macro and math font sizes by setting internal macros \texttt{\_sizemtext}, \texttt{\_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 typosize \ urgedotname \#1/#2\{\%
  \_textfontsize\#1\_mathfontsize\#1\_setbaselineskip\#2\%
  \_setmainvalues _ignorespaces
\} \protected\_def \_typosize \ urgingdotname \#1/#2\{\%
  \_textfontsize\#1\_mathfontsize\#1\_setbaselineskip\#2\%
\}

\texttt{\textbackslash setmainvalues} \_setmainvalues \ sets the current font size and \baselineskip values to the \texttt{\mainfontsize} and \texttt{\mainbaselineskip} registers and loads fonts at given sizes. It redefines itself as \texttt{\_setmainvaluesL} to set the main values only first. The \texttt{\_setmainvaluesL} does only fonts loading. \texttt{\scalemain} returns to these values if they were set. Else they are set to 10/12pt. \texttt{\mfontsrule} gives the rule how math fonts are loaded when \texttt{\typosize} or \texttt{\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.

```
\newskip \mainbaselineskip \mainbaselineskip=0pt \relax
\newdimen \mainfontsize \mainfontsize=0pt
\newcount \mfontsrule \mfontsrule=1
\def \setmainvalues {\mainbaselineskip=\baselineskip \mainfontsize=\optsize
\topskip=\mainfontsize \splittopskip=\topskip
\ifmmode \else \rm \fi % load and initialize \rm variant
\ifnum \mfontsrule>0 \normalmath \fi % load math fonts first
\let \setmainvalues=\setmainvaluesL}
\def \setmainvaluesL {\relax \ifmmode \else \rm \fi % load text font
\ifcase \mfontsrule % load math fonts
\or \ifnum \currentgrouplevel=0 \normalmath \else \everymath=\setmathfonts \everydisplay=\normalmath \fi
\or \normalmath \fi
\or \normalmath \fi
\def \scalemain {\ifdim \mainfontsize=\zo \mainfontsize=10pt \mainbaselineskip=12pt
\let \setmainvalues=\setmainvaluesL
\fi
\optsize=\mainfontsize \baselineskip=\mainbaselineskip}
\public \scalemain \mainfontsize \mainbaselineskip \mfontsrule ;
```

Suppose following example: {\typosize[13/15] Let $M$ be a subset of $R$ 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 process is register 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.

```
\thefontsize [⟨size⟩] and \thefontscale [⟨factor⟩] do modification of the size of the current font. They are implemented by the \newcurrfontsize macro.
```

```
\protected\def \thefontsize[#1]{\if$#1$\else
\tmpdim=#1\ptunit \newcurrfontsize{at\tmpdim}\fi
\ignorespaces}
\protected\def \thefontscale[#1]{\ifx$#1$\else
\tmpdim=#1pt \divide \tmpdim by1000 \tmpdim=\ea\ea\ea\ignorept \pdffontsize\font \tmpdim
\newcurrfontsize{at\tmpdim}\fi
\ignorespaces}
\public \thefontsize \thefontscale ;
```

$\em$ keeps the weight of the current variant and switches roman ↔ italic. It adds the italic correction by the \additcorr and \afteritcorr macros. The second does not add italic correction if the next character is dot or comma.
The \boldify macro does \let\rm\bf, \let\it\bi and \let\normalmath=\boldmath. All following text will be in bold. If should be used after \typosize or \typoscale macros.

The internal \_runboldmath macro runs \_boldmath immediately if no delay of the math font loading is set by \_setmainvaluesL.

The \rm, \it in math mode must keep its original meaning.

We need to use a font selector for default pagination. Because we don’t know what default font size will be selected by the user, we use this \_rmfixed macro. It sets the \rm font from the default font size (declared by first \typosize command and redefines itself be only the font switch for the next pages.

\_begoutput which does:
- increments \_pageno,
- prints \_Xpage{\_pageno} to the .ref file (if \openref is active),
- calculates \_hoffset,
- sets local meaning of macros used in headlines/footlines (see \_regmacro).

\_shipout\_completepage, which is \vbox of –
- background box, if \_pgbackground is non-empty,
- headline box by \_makeheadline, if the \headline is nonempty,
- \vbox to\_vsize of \_pagecontents which consists of –
  - \_pagedest, the page destination \_pg:\{\_pageno\} 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, \fnote),
  - \_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 \_dostereject is activated.

2.18 Output routine

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

- \_begoutput which does:
  - increments \_pageno,
  - prints \_Xpage{\_pageno} to the .ref file (if \openref is active),
  - calculates \_hoffset,
  - sets local meaning of macros used in headlines/footlines (see \_regmacro).
- \_shipout\_completepage, which is \vbox of –
  - background box, if \_pgbackground is non-empty,
  - headline box by \_makeheadline, if the \headline is nonempty,
  - \vbox to\_vsize of \_pagecontents which consists of –
    - \_pagedest, the page destination \_pg:\{\_pageno\} 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, \fnote),
    - \_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 \_dostereject is activated.
\_optexoutput is the default output routine. You can create another...
The \_preshipout\{destination box number\}\{box specification\} used here behaves similarly like \_setbox but it does not only copy the box contents but adds the color literals depending on used attributes. It is defined using lua code, see section 2.39.

```
\_optexoutput
```

Default \_begoutput and \_endoutput is defined. If you need another functionality implemented in the output routine, you can \addto\_begoutput{...} 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.

```
\_begoutput
```

The \hsize value can be changed at various places in the document but we need to have a constant value \_xhsize in the output routine (for headlines and footlines, for instance). This value is set from the current value of \hsize when \_setxhsize macro is called. This macro destroys itself, so the value is set only once. Typically it is done in \_margins macro or when first \_optexoutput routine is called (see \_begoutput). Or it is called at the begining of the \begtt\...\endtt environment before \hsize value is eventually changed by the user in this environment.

```
\_setxhsize
```

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

```
\_makeheadline
```

The \_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.

\_def \_pagecontents{\_pagedest % destination of the page
\_ifvoid \_topins \_else \_unvbox \_topins \_fi
\_dimen0=\dp255 \_unvbox255 % open up \box255
\_ifvoid \_footins \_else \footnote info is present
\_vskip \_skip \_footins
\_footnoterule \_unvbox \_footins \_fi
\_kern-\_dimen0 \_vskip \_pgbottomskip}

\_def \_pagedest {{\_def \_destheight{25pt} \_dest[pg:\the \_pageno]}}

\_def \_footnoterule {\_kern-3pt \_hrule width 2truein \_kern 2.6pt }

\pageno, \folio, \nopagenumbers, \advancepageno and \normalbottom used in the context of the output routine from plain TeX is defined here. Only the \raggedbottom macro is defined differently. We use the \pgbottomskip register here which is set to 0pt by default.

\_countdef \_pageno=0 \_pageno=1 % first page is number 1
\_def \_folio {\_ifnum \_pageno<0 \_romannumeral-\_pageno \_else \_number \_pageno \_fi}
\_def \_nopagenumbers {\_footline={}}
\_def \_advancepageno {\_ifnum \_pageno<0 \_decr \_pageno \_else \_incr \_pageno \_fi}
\_def \_raggedbottom {\_topskip=\_dimexpr \_topskip plus60pt \_pgbottomskip=0pt plus1fil \_relax}
\_def \_normalbottom {\_topskip=\_dimexpr \_topskip \_pgbottomskip=0pt \_relax}

Macros for footnotes are the same as in plain TeX. There is only one difference: \vfootnote is implemented as \_opfootnote with empty parameter #1. This parameter should do local settings inside the \footins group and it does it when \fnote macro is used.
The \_opfootnote and \vfootnote don't take the footnote text as a parameter. This is due to a user can do catcode settings (like inline verbatim) in the footnote text. This idea is adapted from plain TeX. The \footnote and \footstrut is defined as in plain TeX.

The \topins macros \topinsert, \midinsert, \pageinsert, \endinsert are the same as in plain TeX.
\newif\ifupage \newif\ifumid
\def \topinsert {\umidfalse \upagetrue \oins}
\def \midinsert {\umidtrue \oins}
\def \pageinsert {\umidfalse \upagetrue \oins}
\skip_topins = \zoskip % no space added when a \topinsert is present
\count_topins = 1000 % magnification factor (1 to 1)
\dimen_topins = \maxdimen % no limit per page
\parskip = \zoskip % start a \vbox
\parskip = \zoskip % finish the \vbox
\ifumid \dimen0 = \ht0 \advance \dimen0 by \dp0 \advance \dimen0 by \baselineskip
\advance \dimen0 by \pagetotal \advance \dimen0 by - \pageshrink
\ifdim \dimen0 > \pagegoal \umidfalse \upagetrue \fi \fi
\ifumid \bigskip \box0 \bigbreak \else \insert \topins \penalty100 % floating insertion
\skiptopskip = 0pt \splitmaxdepth = \maxdimen \floatingpenalty = 0
\ifupage \dimen0 = \dp0 \vbox to \vsize {\unvbox0 \kern - \dimen0} % depth is zero
\else \box0 \nobreak \bigskip \fi \fi \endgroup
\public \topins \topinsert \midinsert \pageinsert \endinsert ;

The \draft macro is an example of usage \pgbackground to create watercolor marks.

\public \draft ;

2.19 Margins

The \margins macro is documented in the section 1.2.1.

\margins_MACRO {Macros for margins setting <2021-03-15>} % preloaded in format

\margins_MACRO 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.

\public \margins ;

\def \margins #1 #2 (#3,#4,#5,#6)#7 {\def \tmp{#7} % \ifx \tmp \empty \opwarning {\string \margins: missing unit, mm inserted} \def \tmp{mm} \fi
\setpagedimens #2 % setting \pgwidth, \pgheight
\ifdim \pgwidth = 0pt \else \hoffset = 0pt \voffset = 0pt \fi \if#3 \if#4 \hoffset = \dimexpr ( \pgwidth - \hsize)/2 \relax \else \hoffset = \dimexpr \pgwidth - \hsize - #4 \tmp \relax \fi \fi \else \if#4 \hoffset = #3 \tmp \relax \else \hsize = \dimexpr \pgwidth - \hsize - #4 \tmp \relax \fi \fi \hoffset = \dimexpr \pgwidth - \hsize \relax % only right margin
\else \if#4 \hoffset = #3 \tmp \relax % only left margin \else \hsize = \dimexpr \pgwidth - \hsize - #3 \tmp - #4 \tmp \relax \fi \fi \hsize = \dimexpr \pgwidth - \hsize \relax % left+right margin
\else \if#4 \hoffset = #3 \tmp \relax \else \hsize = \dimexpr \pgwidth - \hsize \relax \fi \fi \xhsize = \hsize \setxhsize \xhsize used by \output routine
\else \if#4 \hoffset = #3 \tmp \relax \else \hsize = \dimexpr \pgwidth - \hsize \relax \fi \fi \voffset = \dimexpr \vsize/2 \relax \else \voffset = \dimexpr \vsize - \hsize \relax \fi \fi \voffset = \dimexpr \vsize - \hsize \relax % only bottom margin
The common page dimensions are defined here.

\public \margins ;

\magscale \[\langle \text{factor} \rangle \] does \texttt{mag=}\langle \text{factor} \rangle and recalculates page dimensions to their true values.

\public \margins ;

\magscale [(factor)] does \texttt{mag=}factor and recalculates page dimensions to their true values.

\pdfpagewidth=\pgwidth \pdfpageheight=\pgheight

\public \magscale ;

\magscale [(factor)] does \texttt{mag=}factor and recalculates page dimensions to their true values.

\public \margins ;

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 a 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 depend on pages) and color handling in PDF.

TEX itself has no concept of colors. Colors have always been handled by inserting whatsitns (either using \texttt{\special} for DVI or using \texttt{\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 PDF attributes to LuaTEX attributes in version 1.04 of OpTEX. Using attributes, the color setting behaves exactly like font selection from TEX point of view: it respects TEX groups, colors can span more pages, independent colors can be set for \texttt{\insert}s, etc. Moreover, once a material is created (using \texttt{\setbox} for example) then it has its fonts and its colors frozen and you can rely on it when you are using e.g. \texttt{\unhbox}. There are no internal whatsitns for colors which can interfere with other typesetting material. In the end something like setting text to red (\texttt{\textcolor{red}{Red text}}) should have the same nice behavior like setting text to bold (\texttt{\textbf{bf text}}).

LuaTEX attributes can be set like count register – one attribute holds one number at a time. The value of attribute is propagated to each created typesetting element until the attribute is unset or set to another value. Very much like the font property. We use one attribute \texttt{\_colorattr} for storing the currently selected color (in number form).
Macros \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 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.

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 \_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 subtracted in such case and the negative components are rounded to zero immediately. For example

\colordef \Color {\Brown-\Black}

can be used for removing the black component from the color. You can use the -\Black trick after \useK command to remove grey components occurred during color mixing.

Finally, you can use "^" immediately 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 basic colors in CMYK \Blue \Red \Brown \Green \Yellow \Cyan \Magenta \Grey \LightGrey \White and \Black are declared here.
By default, the \setcmykcolor \setrgbcolor and \setgreycolor macros with \{⟨color-data⟩\} parameter expand to \_setcolor{⟨color-data⟩}{⟨fill-op⟩}{⟨stroke-op⟩} where \{⟨color-data⟩\} is \{R⟩ ⟨G⟩ ⟨B⟩ or \{C⟩ ⟨M⟩ ⟨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⟩
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.

```
\protected\def \_localcolor {\_let \_colorprefix = \_relax}
\protected\def \_nolocalcolor {\_let \_colorprefix = \global}
\public \_localcolor \_nolocalcolor ;
```

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⟩⟩}{<</ca X /CA X>>}` is applied (where `X` is `(255-⟨number⟩)/255`). This information is used when chipout is processed (similarly as colors). It means `/tr⟨⟨number⟩⟩` gs is inserted when the attribute is changed.

```
\_protected\def \_resetattrs {\_colorattr = \noattr \_transpattr = \noattr}
\_public \_resetattrs ;
```

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.

The \rgbcolordef and \cmykcolordef use common macro \_commoncolordef with different first four parameters. The \_commoncolordef (selector)\{\textit{K}\}\{\textit{R}\}\{\textit{G}\}\{\textit{B}\}\{\textit{Y}\}\{\textit{K}\} does the real work. It initializes the Lua variables for summation. It expands (\textit{data}) in the group where color selectors have special meaning, then it adjusts the resulting string by \replstring and runs it. Example shows how the (\textit{data}) are processed:

```
\textbf{input (\textit{data}):} \texttt{".3\textbackslash{}Blue + .6\textbackslash{}KhakiC \textbackslash{}useK \textbackslash{}Black"}
\textbf{expanded to:} \texttt{".3 \textbackslash{}!\textbackslash{}K 1 1 0 0 + .6\textbackslash{}!\textbackslash{}R .804 .776 .45 \_\textbackslash{}useK \textbackslash{}=!\textbackslash{}G 0"}
\textbf{adjusted to:} \texttt{"\_\textbackslash{}addcolor \_\textbackslash{}3!\textbackslash{}!\textbackslash{}K 1 1 0 0 \_\textbackslash{}addcolor \_\textbackslash{}6!\textbackslash{}!\textbackslash{}R .804 .776 .45 \_\textbackslash{}useK \_\textbackslash{}addcolor -1!\textbackslash{}!\textbackslash{}G 0"}
\textbf{and this is processed.}
```

\_\textbackslash{}addcolor (\textit{coef})\textbackslash{}!\{\textit{mod}\} (\textit{type}) expands to \_\textbackslash{}addcolor: (\textit{mod}\} (\textit{type}) (\textit{coef}) for example it expands to \_\textbackslash{}addcolor: \_\textbackslash{}addcolor: \_\textbackslash{}addcolor: (\textit{addcolor:} \_\textbackslash{}addcolor: \_\textbackslash{}addcolor: (\textit{type}) (\textit{coef}) followed by one or three or four numbers (depending on \textit{type}). (\textit{mod}) is = \{\textit{use as is}\} or " \{\textit{use complementary color}\}. (\textit{type}) is \textit{K} for CMYK, \textit{R} for RGB and \textit{G} for GREY color space. Uppercased (\textit{type}) informs that \cmykcolordef is processed and lowercased (\textit{type}) informs that \rgbcolordef is processed. All variants of commands \_\textbackslash{}addcolor: (\textit{mod}\} (\textit{type}) are defined. All of them expand to \textbf{\_\textbackslash{}addcolorA (\textit{eft}) \{\textit{v2}\} \{\textit{v3}\} \{\textit{v4}\} which adds the values of Lua variables. The \rgbcolordef uses \_\textbackslash{}addcolorA (\textit{R}) \{\textit{G}\} \{\textit{B}\} \{\textit{Y}\} \{\textit{K}\} and \cmykcolordef uses \_\textbackslash{}addcolorA (\textit{C}) \{\textit{M}\} \{\textit{Y}\} \{\textit{K}\}. So the Lua variable names are a little confusing when \rgbcolordef is processed.

Next, \_\textbackslash{}commoncolordef saves resulting values from Lua to \_\textbackslash{}tmpb using \colordefFin. If \rgbcolordef is processed, then we must to remove the last \textit{\textbackslash{}K} component which is in the format .0 in such case. The \_\textbackslash{}stripK macro does it. Finally, the \textit{what-define} is defined as (\textit{selector})\{\textit{expanded _\textbackslash{}tmpb}\}, for example \_\textbackslash{}setcmykcolor\{1 0 .5 .3\}.

```
\textbf{colors.omn}
```
\def\setrgbcolor##1{\ifnum##1=0 \relax \else \edef\tmpb{##1}\replstring\tmpb{-}{\_addcolor} \replstring\tmpb{+}{\_addcolor}\_replstring\tmpb{^}{!}\replstring\tmpb{-!}{!}\_ifx \useK \relax \_edef\tmpb{\colordefFin};\fi \_tmpb \_edef\tmpb{\_colordefFin} \_ifx \_useK \relax \_edef\tmpb{\_ea\_stripK \_tmpb;}\fi \_ea\_endgroup \_ea\def\ea#5\ea{\_ea#1\ea{\_tmpb}} \sdef{\_addcolor:=K}##1##2##3{\_cs{\_addcolor##2##3}##1} \sdef{\_addcolor:^K}##1##2##3##4##5{\_cs{\_addcolor##2##3##4##5}##1\_addcolorA ##1##2##3##4##5} \sdef{\_addcolor:^G}##1##2{\_addcolorA ##10000\_addcolorA ##1##2000} \sdef{\_addcolor:^R}##1##2##3##4{\_cs{\_addcolor##2##3##4##5}##1\_addcolorA ##1##2##3##4##5\_addcolorA ##1##2##3##4##5} \sdef{\_addcolor:^k}##1##2##3##4##5{\_cs{\_addcolor##2##3##4##5}##1\_addcolorA ##1##2##3##4##5\_addcolorA ##1##2##3##4##5} \sdef{\_addcolor:^g}##1##2{\_addcolorA ##1##2##3##4##5\_addcolorA ##1##2##3##4##5} \sdef{\_addcolor:^r}##1##2##3##4{\_addcolorA ##1##2##3##4##5\_addcolorA ##1##2##3##4##5} \sdef{\_stripK}##1.0;{##1} \let\_colordef=\cmykcolordef % default \_colordef is \_cmykcolordef

Public versions of \_colordef and \_useK macros are declared using \_def, because the internal versions \_colordef and \_useK are changed during processing.

colors.opm

The \LaTeX file x11nam.def is read by \morecolors. The numbers 0,1,2,3,4 are transformed to letters O, ⟨none⟩, B, C, D in the name of the color. Colors defined already are not re-defined. The empty \_showcolor macro should be re-defined for color catalog printing. For example:

\def\vr{\vrule height10pt depth2pt width20pt} \def\_showcolor{\hbox{\tt\_bslash\_tmpb: \csname\_tmpb\endcsname \vr}\space\space} \begmulti 4 \typosize[10/14] \morecolors \endmulti

colors.opm

The \LaTeX file x11nam.def is read by \morecolors. The numbers 0,1,2,3,4 are transformed to letters O, ⟨none⟩, B, C, D in the name of the color. Colors defined already are not re-defined. The empty \_showcolor macro should be re-defined for color catalog printing. For example:

\def\vr{\vrule height10pt depth2pt width20pt} \def\_showcolor{\hbox{\tt\bash\_tmpb: \csname\_tmpb\endcsname \vr}\space\space} \begmulti 4 \typosize[10/14] \morecolors \endmulti

colors.opm
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.

```
\_newwrite\reffile
\_def\_inputref {%
  \_isfile{\jobname.ref}\_iftrue
  \_input {\jobname.ref}%
  \_edef\prevrefhash{\_mdfive{\jobname.ref}}%
  \_gfnotenum=0 \_lfnotenum=0 \_mnotenum=0
  \_openref
  \_fi
  }
}
```

\_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.

```
\_def\_mdfive#1{\_directlua{optex.mdfive(#1)}}
\_def\_prevrefhash()
```

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.

```
\_def\_wrefrelax\_wrefrelax{}
\_let\_wref=\_wrefrelax
```
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{⟨macro⟩}{⟨data⟩} so that it saves the line ⟨(macro)(data)⟩ to the .ref file using the asynchronous \write primitive. Finally, \openref destroys itself, because we don’t need to open the file again.

\_wref{⟨csname⟩}{⟨params⟩} in fact does \write\_reffile{⟨csname⟩}{⟨params⟩} and similarly \_ewref{⟨csname⟩}{⟨params⟩} does \write\_reffile{⟨csname⟩}{⟨expanded-params⟩}.

We are using the convention that the macros used in .ref file are named \_X{⟨foo⟩}. 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{⟨version⟩}

The macro checks the version compatibility. Because OPmac does not understand \Xrefversion we use \Xrefversion (with a different number of ⟨version⟩ 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{⟨definitions of your ref macros⟩}. This command writes ⟨definitions of your ref macros⟩ 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 ⟨definitions of your ref macros⟩ 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
\_def\_unresolvedrefs#1{0}

\_Xlabel \{\langle label \rangle\} \{\langle given-text \rangle\} saves the \langle given-text \rangle to \_lab: \langle label \rangle and saves \{\langle gpageno \rangle\}{\langle pageno \rangle\} to \_pgref: \langle label \rangle.

\_Xlabel \{\langle label \rangle\} \{\langle text \rangle\} saves the \langle text \rangle into \_currpage. Resets \_lfnotenum; it is used if footnotes are numbered from one at each page.

\_def\_Xpage#1#2{\_def\_currpage{{#1}{#2}}\_lfnotenum=0}

\_lastreflabel keeps the value of the last label read by \ref or \pgref. You can use it for example by defining a macro \pg by \def\pg{\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 \_pg.

\_addtocounter{\unresolvedrefs}{1}
\_warning{There are \_the\_unresolvedrefs\_space unresolved references}
\_unresolvedrefs=0

\_def\_lastlabel[#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
\_ignorespaces
\_let \_slideshook=\_relax % redefined if \slides + \slideshow.
\_def\_wlabel#1{\_ifx\_lastlabel\_undefined \_else \_dest[ref:\_lastlabel]{\_reftext{\csname l0:\_lastlabel\endcsname}{#1}}\_ewref\_Xlabel\{\_lastlabel\}{\_lastreflabel\}}
\_def\_ref#1{\_xdef\_lastreflabel{#1}\_isnextchar\bgroup{\_refA}{\_refA{@}}}
\_def\_refA#1{\_isdefined{\_lab:\_lastreflabel}\_iftrue \_ilink[ref:\_lastreflabel]{\_reftext{\csname _lab:\_lastreflabel\_endcsname\}{#1}}\_else \_reftext{??}{#1}\_opwarning{label \_lastreflabel unknown. Try to TeX me again}\_incr\_unresolvedrefs \_openref\_fi\_fi}
\_def\_pgref#1{\_xdef\_lastreflabel{#1}\_isnextchar\bgroup{\_pgrefA}{\_pgrefA{@}}}
\_def\_pgrefA#1{\_isdefined{\_pgref:\_lastreflabel}\_iftrue \_ea\_ea\_ea\_pgrefB{\csname _pgref:\_lastreflabel\endcsname\}{#1}}\_else \_reftext{??}{#1}\_opwarning{pg-label \_lastreflabel unknown. Try to TeX me again}\_incr\_unresolvedrefs \_openref\_fi\_fi
\_def\_pgrefB#1{\_ilink[pg:#1]{\_reftext{\_csname pg:\_lastreflabel\_endcsname\}{#1}}}
\_def\_ref{\ref \pgref ;}

\_def\_Xpage{\{\langle gpageno \rangle\}{\langle pageno \rangle\} saves the parameter pair into \_currpage. Resets \_lfnotenum; it is used if footnotes are numbered from one at each page.
\_reftext{\{implicit-text\}\{given-text\}} expands to the \{given-text\} but the optional \_ in the \{given-text\} is replaced by the \{implicit-text\} first.

Each hyperlink is created internally by \_xlink{\{type\}\{spec\}\{color\}\{text\}}. This macro expands to \_xlink{\{type\}\{spec\}\{color\}\{text\}} by default, i.e. no active hyperlink is created, only \{text\} is printed in horizontal mode (and in a group). If \_hyperlinks is used, then \_xlink gets the meaning of \_xlinkactive and

\_xlinkactive{\{type\}\{spec\}\{color\}\{text\}} expands to \_xlink{\{type\}\{spec\}\{color\}\{text\}} by default, i.e. no active hyperlink is created, only \{text\} is printed in horizontal mode (and in a group). If \_hyperlinks is used, then \_xlinkactive gets the meaning of \_xlinkactive and

2.23 Hyperlinks

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

- ref:\{label\} – the destination is created when \_label{\{label\}} is used, see also the section 2.22.
- toc:\{tocrefnum\} – the destination is created at chap/sec/sec names, see also the section 2.24.
- pg:\{gpageno\} – the destination is created at beginning of each page, see also the section 2.18.
- cite:\{bibpart\}\{bibnum\} – the destination is created in bibliography reference, see section 2.32.1.
- fn:\{fnf\} – link from footnote to text, see also section 2.34.
- fn:\{fnf\} – link form text to footnote, see also section 2.34.
- url:\{url\} – used by \url or \ulink, see also the end of this section.

The \{tocrefnum\}, \{gpageno\}, \{bibnum\}, and \{fnnum\} are numbers starting from one and globally incremented by one in the whole document. The registers \_tocrefnum, \_gpageno, \_bibnum, and \_fnnum are used for these numbers.

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

The color for active links can be declared by \def\_linkcolor{\{color\}}, the border around link can be declared by \_linkcolor{\{type\}\{spec\}}. These macros are not declared by default, so color for active links are given only by \_hyperlinks macro and borders are invisible. For example \def\_linkcolor{\Red} means that links from table of contents are in red. Another example \def\_linkcolor{\Red} causes red frames in TOC (not printed, only visible in PDF viewers).

\dest{\{type\}\{spec\}} creates a destination of internal links. The destination is declared in the format \{type\}\{spec\}. If the \_hyperlinks command in not used, then \dest does nothing else it is set to \_destactive. The \_destactive is implemented by \_pdfdest primitive. It creates a box in which the destination is shifted by \_destheight. The reason is that the destination is exactly at the top border of the PDF viewer but we want to see the line where the destination is. The destination box is positioned by a different way dependent on the current vertical or horizontal mode.

Each hyperlink is created internally by \_xlink{\{type\}\{spec\}\{color\}\{text\}}. This macro expands to \_xlink{\{type\}\{spec\}\{color\}\{text\}} by default, i.e. no active hyperlink is created, only \{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 \langle text\rangle has given \langle color\rangle only when hyperlink is created. If \langle type\rangle is defined, it has precedence over \langle color\rangle.

The \_linkdimens macro defines the dimensions of link area.

A specific action can be defined for each link \langle type\rangle by the macro \_action\{\langle spec\rangle\}. OpTeX\n\text{defines only} \_action\{\langle url\rangle\}. The default link action (when \_action\{\langle spec\rangle\} is not defined) is \goto name{\langle type\rangle}{\langle spec\rangle} (an internal link). It is declared in the \_linkactions\{\langle type\rangle\}{\langle spec\rangle}\} macro.

The \pdfstartlink primitive uses \attr{\pdfborder\{\langle type\rangle\}}\}. The \pdfborder\{\langle type\rangle\}\} macro expands to /C[? ? ?] \Border[0 0 .6] if the \_border macro (i.e. \refborder, \citeborder, \tocborder, \gborder, \urlborder, \fntborder or \fntfborder) is defined.

\let\_linkcolor\#1\empty
\let\_ilinkcolor\#1\empty
\let\_dest=\_destactive \_let\_xlink=\_xlinkactive
\let\_ilinkcolor=\#1\empty
\let\_xlinkcolor=\#1\empty
\public \dest \xlink \link \%
\public \hyperlinks \url \ulink \%
\public \hyperlinks\#1\%
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}
\let\urlxskip=\nobreak
\def\url#1{{% 
\def\tmpa{#1}\replstring\tmpa {\|}{}}% 
\def\tmpb{#1}\replstring\tmpb {||}{||}% 
{\escapechar=-1 \ea}\edef\ea\tmpb\ea{\detokenize\ea{\tmpb}}% 
\replstring\tmpb{||}{gb|}% 
\replstring\tmpb{ }{ }% 
\replstring\tmpb{://}{://}% 
{\ea\ulink\ea{\ea{\tmpa}}} {\urlfont\ea\urlA\tmpb\ea\fin}% 
}}% 
\def\urlA#1{\ifx\fin#1\else \urlC{}{#1}\fi}% 
\def\urlB#1{\ifx\fin#1\else \urlC{\urlxskip}{#1}\fi}% 
\def\urlC#1#2{% 
\ifcsname _ur:#2\endcsname \lastnamedcs\urlA\fi\urlB% 
}% 
\sdef{_ur:://}{\urlskip:/\urlskip:/\urlbskip}% 
\sdef{_ur:/}{\urlskip:/\urlbskip}% 
\sdef{_ur:.}{\urlskip:.\urlbskip}% 
\sdef{_ur:?}{\urlskip:?\urlbskip}% 
\sdef{_ur:=}{\urlskip:=\urlbskip}% 
\sdef{_ur:-}{\urlskip:-\urlbskip}% 
\sdef{_ur:&}{\urlskip&\urlbskip}% 
\sdef{_ur:gb|}{\urlgskip}% 
\edef\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 \\}
\let\public\url ;
\end{verbatim}

2.24 Making table of contents

\begin{verbatim}
\Xtoc{\level}{\type}{\number}{\o-title}{\title} \textup{(in .ref file)} reads given data and appends them to the \_toclist as \_tocline{\level}{\type}{\number}{\o-title}{\title}{\gpageno}{\pageno} where:

- \level: 0 reserved, 1: chapter, 2: section, 3: subsection
- \type: the type of the level, i.e. chap, sec, secc
- \number: the number of the chapter/section/subsection in the format 1.2.3
- \o-title: outlines title, if differs from the \title.
- \title: the title text
- \gpageno: the page number numbered from 1 independently of pagination
- \pageno: the page number used in the pagination

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}
To 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:

```
\llap{⟨number⟩} text text text text text text 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).

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 whith rapped ⟨pageno⟩.
- \_pgn{⟨pageno⟩} creates ⟨pageno⟩ as link to real ⟨gpage⟩ saved in #6 of \_tocline. This is temporarily defined in the \_tocline.

If you want a special formating 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:

```
def\addtotoc{⟨level⟩}{⟨type⟩}{⟨number⟩}{⟨o-title⟩}{⟨title⟩}\
\incr\_tocrefnum\dest[\toc:\_the\_tocrefnum]\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):

```
def\_tocl:10\dest[\toc:\_the\_tocrefnum]\ewref_Xtoc{⟨#1⟩}{⟨#2⟩}{⟨#3⟩}{⟨#4⟩}{⟨#5⟩}%
```

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.opm

\_def\_maketoc\_(\_par \_ifx\_toclist\_empty
\_opwarning\_(\_noexpand\maketoc -- data unavailable, TeX me again)\_openref
\_incr\_unresolvedrefs
\_else \_beginingroup
\_tocrefnum=0 \_penalty11333
\_the\_regtoc \_toclist
\_endgroup \_fi
\}

\regmacro appends its parameters to \_regtoc, \_regmark and \_regoul. These token lists are used in \maketoc, \_begoutput and \pdfunidef.

\_newtoks \_regtoc \_newtoks \_regmark \_newtoks \_regoul
\_def\_regmacro #1#2#3{\_toksapp\_regtoc{#1}\_toksapp\_regmark{#2}\_toksapp\_regoul{#3}}
\_public \maketoc \regmacro ;

\_newtoks \_regtoc \_newtoks \_regmark \_newtoks \_regoul
\_def\_regmacro #1#2#3{%\_toksapp\_regtoc{#1}\_toksapp\_regmark{#2}\_toksapp\_regoul{#3}%
\}
\_public \maketoc \regmacro ;

\_codedecl \outlines {PDF outlines <2021-02-09>} % preloaded in format
\_def\_outlines#1{\_pdfcatalog{/PageMode/UseOutlines}\_openref
\_ifx\_toclist\_empty
\_opwarning\_(\_noexpand\outlines -- data unavailable. TeX me again)\_openref
\_incr\_unresolvedrefs
\_else
\_beginingroup
\_tocrefnum=0 \_penalty11333
\_the\_regtoc \_toclist
\_endgroup
\}

\_newtoks \_regtoc \_newtoks \_regmark \_newtoks \_regoul
\_def\_regmacro #1#2#3{%\_toksapp\_regtoc{#1}\_toksapp\_regmark{#2}\_toksapp\_regoul{#3}%
\}
\_public \maketoc \regmacro ;

\_codedecl \outlines {PDF outlines <2021-02-09>} % preloaded in format
\_def\_outlines#1{\_pdfcatalog{/PageMode/UseOutlines}\_openref
\_ifx\_toclist\_empty
\_opwarning\_(\_noexpand\outlines -- data unavailable. TeX me again)\_openref
\_incr\_unresolvedrefs
\_else
\_beginingroup
\_tocrefnum=0 \_penalty11333
\_the\_regtoc \_toclist
\_endgroup
\}

2.25 PDF outlines

2.25.1 Nesting PDF outlines

The problem is that PDF format needs to know the number of direct descendants of each outline if we need to create the tree of structured outlines. But we know only the level of each outline. The required data should be calculated from TOC data. We use two steps over TOC data saved in the \_toclist where each record is represented by one \_tocline.

The first step, the \_outlines macro sets \_tocline to \_outlinesA and calculates the number of direct descendants of each record. The second step, the \_outlines macro sets \_tocline to \_outlinesB and it uses prepared data and creates outlines.

Each outline is mapped to the control sequence of the type \_ol:⟨num⟩ or \_ol:⟨num⟩:⟨num⟩ or \_ol:⟨num⟩:⟨num⟩:⟨num⟩ etc. The first one is reserved for level 0, the second one for level 1 (chapters), the third one for level 2 (sections) etc. The number of direct descendants will be stored in these macros after the first step is finished. Each new outline of a given level increases the ⟨num⟩ at the given level. When the first step is processed then (above that) the \_ol:... sequence of the parent increases its value too. The \_ol:... sequences are implemented by \_ol:\_count0:\_count1:\_count2 etc. For example, when section (level 2) is processed in the first step then we do:

\_advance \_count2 by 1
\_advance \_ol:\_count0:\_count1:\_count2 of this section by 1

\_count2 \_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.
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 

\begin{verbatim}
<FEFF 0043 0076 0069 010D 0065 006E 00ED 0020 006A 0065 0020 007A 00E1 0074 011B 017E 0082 D835DD44>
\end{verbatim}

This example shows a hexadecimal PDF string (enclosed in \verb|<| as opposed to the literal PDF string enclosed in \verb|(|. 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” \verb|FEFF|. Each unicode character is then encoded in one or two byte pairs. The example string corresponds to the text “Cvičení je zátěž a x ∈ 𝕄”.

pdfuni-string.opm

\begin{verbatim}
\_ifx\_dest\_destactive \_else
\_opwarning{\noexpand\outlines doesn't work when \noexpand\hyperlinks isn't declared}\_fi
\{\_let\_tocline=\_outlinesA
\_def\_outlinenivel(#1)\{\_let\_tocline=\_outlinesB
\_tocrefnum=0 \_count0=0 \_count2=0 \_count3=0
\_toclist \% calculate numbers o childs
\_def\_outlinelevel{#1}\_let\_tocline=\_outlinesB
\_tocrefnum=0 \_count0=0 \_count2=0 \_count3=0
\_toclist \% create outlines
\_fi
\}
\_def\_outlinesA#1#2#3#4#5#6#7{\%
\_isequal{\relax}{#4}\_iffalse
\_advance\_count#1 by1
\_ifcase#1\_or
\_addoneol{_ol:\_the\_count0}\_or
\_addoneol{_ol:\_the\_count0:\_the\_count1}\_or
\_addoneol{_ol:\_the\_count0:\_the\_count1:\_the\_count2}\_or
\_addoneol{_ol:\_the\_count0:\_the\_count1:\_the\_count2:\_the\_count3}\_fi
\_fi
\}
\_def\_addoneol#1{\%
\_ifcsname #1\_endcsname
\_tmpnum=\_csname#1\_endcsname\relax
\_advance\_tmpnum by1 \_sxdef{#1}{\_the\_tmpnum}\%
\_else \_sxdef{#1}{1}\%
\_fi
\}
\_def\_outlinesB#1#2#3#4#5#6#7{\%
\_advance\_tocrefnum by1
\_isequal{\relax}{#4}\_iffalse
\_advance\_count#1 by1
\_ifcase#1\%
\_tmpnum=\_trycs{_ol:\_the\_count0}{0}\_or
\_tmpnum=\_trycs{_ol:\_the\_count0:\_the\_count1}{0}\_relax\_or
\_tmpnum=\_trycs{_ol:\_the\_count0:\_the\_count1:\_the\_count2}{0}\_relax\_or
\_tmpnum=\_trycs{_ol:\_the\_count0:\_the\_count1:\_the\_count2:\_the\_count3}{0}\_relax\_or
\_tmpnum = 0\_relax\_fi
\_isempty{#4}\_iftrue \_pdfunidef\_tmp{#5}\_else \_pdfunidef\_tmp{#4}\_fi
\_outlinesC{toc:\_the\_tocrefnum}{\_ifnum#1<\_outlinelevel\_space\_else-}\_fi{\_tmpnum}{\_tmp}\%
\_fi
\}
\_def\_outlinesC#1#2#3#4{\_pdfoutline goto name{#1} count #2#3{#4}\_relax}
\_newcount\_oulnum
\_def\_insertoutline#1{\_incr\_oulnum
\_pdfdest name{oul:\_the\_oulnum} xyz\_relax
\_pdfunidef\_tmp{#1}\%
\_pdfoutline goto name{oul:\_the\_oulnum} count0 {\_tmp}\_relax
}\%
\_public \outlines \insertoutline ;
\end{verbatim}
\_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.

\pdfunidef\macro{{\text}} defines \macro as \{text\} converted to Big Endian UTF-16 and enclosed to <>.

Characters for quotes (and separators for quotes) are activated by first \_scantextokens and they are defined as the same non-active characters. But \_regoul can change this definition.

\_prepinverb\{macro\}(\{separator\})\{\text\}, e.g. \_prepinverb\tmkp{aaa \bbb \ccc \dd \ee} does \def\tmkp{\{\prefix \bbb\{\ccc \dd\} \{ \ee\}} where \prefix is \_scantextokens\_unexpanded. It means that in-line verbatim are not argument of \_scantextoken. First \edef\tmkp tokenizes again the \{\text\} but not the parts which were in the the in-line verbatim.
The \regmacro is used in order to set the values of macros \em, \rm, \bf, \it, \bi, \tt, \verb and - to values usable in PDF outlines.

2.26 Chapters, sections, subsections

We are using scaled fonts for titles \titfont, \chapfont, \seccfont and \seccfont. They are scaled from main fonts size of the document, which is declared by first \typosize command.

The \tit macro is defined using \vsctoeol and \printtit. It means that the parameter is separated by end of line and inline verbatim is allowed. The same principle is used in the \chap, \sec, and \secc macros.

You can re-define \printchap, \printsec or \printsecc macros if another design of section titles is needed. These macros get the (title) text in its parameter. The common recommendations for these macros are:

- Use \abovetitle scriptA)\{\{skipA\}\} and \belovetitle scriptB) for inserting vertical material above and below the section title. The arguments of these macros are normally used, i.e. \abovetitle inserts \scriptA\{\{skipA\}\} and \belovetitle inserts \scriptB). But there is an exception: if \belovetitle\{\{skipB\}\} is immediately followed by \abovetitle\{\{penaltyA\}\{\{skipA\}\} (for example section title is immediately followed by subsection title), then only \scriptA\{\{skipA\}\} is generated, i.e. \scriptB\{\{penaltyA\}\{\{skipA\}\} is reduced only to \scriptA\{\{skipA\}\}. 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, \abovetitle\{\{penaltyA\}\{\{skipA\}\} takes previous whatever vertical skip (other than from \belovetitle) and generates only greater from this pair of skips. It means that \{what-ever-skip\}\{\{penaltyA\}\{\{skipA\}\} is transformed to \{penaltyA\}\{\{what-ever-skip\}\{\{skipA\}\}. The reason for such behavior: we don't want to duplicate vertical skips (from \belovetitle) for example above the title.

- Use \printrefnum scriptA)\{\post\}\}\{\prev\}\{\refnum\}\{\post\}. The \refnum is \thechapnum or \thesectionnum or \thesectionnum depending on what type of title is processed. If \nonum prefix is used then \printrefnum prints nothing. The macro \printrefnum does more work: it creates destination of hyperlinks (if \hyperlinks\{\} is used) and saves references from the label (if \label\{\{label\}\} precedes) and saves references for the table of contents (if \maketoc is used).

- Use \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 \TeX trick 0033)

The \chap initializes counters used in chapters, the \sec initializes counters in sections and \secc initializes counters in subsections. If you have more types of numbered objects in your document then you can declare appropriate counters and do \addto\chap\{\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 \chap}.

Default concept: Tables, figures, and display maths are numbered from one in each section – subsections don’t reset these counters. Footnotes declared by \fnotenumchapters are numbered in each chapter from one.

The \the macros \_thechapnum, \_thesecnum, \_theseccnum, \_thetnum, \_thefnum and \_thednum include the format of numbers used when the object is printing. If chapter is never used in the document then \_chapnum=0 and \_othechapnum. expands to empty. Sections have numbers ⟨num⟩ and subsections ⟨num⟩. ⟨num⟩. On the other hand, if chapter is used in the document then \_chapnum>0 and sections have numbers ⟨num⟩. ⟨num⟩. ⟨num⟩ and subsections have numbers ⟨num⟩. ⟨num⟩. ⟨num⟩. ⟨num⟩.
The `\notoc` and `\nonum` prefixes are implemented by internal `\_ifnotoc` and `\_ifnonum`. They are reset after each chapter/section/subsection by the `\_resetnonumnotoc` macro.

```
\newifi \_ifnotoc \_notocfalse \_def \_notoc {\_global\_notoctrue}
\newifi \_ifnonum \_nonumfalse \_def \_nonum {\_global\_nonumtrue}
\def \_resetnonumnotoc {\_global\_notocfalse \_global\_nonumfalse}
\public \notoc \nonum;
```

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.

```
\optdef\_chap[\]{\_trylabel \_scantoeol \_inchap}
\optdef\_sec[\]{\_trylabel \_scantoeol \_insec}
\optdef\_secc[\]{\_trylabel \_scantoeol \_insecc}
\def \_trylabel {\_istoksempty \_opt \_iffalse \_label[\_the\_opt]\_fi}
\def \_inchap #1{\_par \_sectionlevel=1
\_def \_savedtitle {#1}% saved to .ref file
\_ifnonum \_else \_globaldefs=1 \_incr\_chapnum \_chapx\_fi
\_edef \_therefnum {\_ifnonum \_space \_else \_thechapnum \_fi}%
\_printchap{\_scantextokens{#1}}%
\_resetnonumnotoc}
\def \_insec #1{\_par \_sectionlevel=2
\_def \_savedtitle {#1}% saved to .ref file
\_ifnonum \_else \_globaldefs=1 \_incr\_secnum \_secx\_fi
\_edef \_therefnum {\_ifnonum \_space \_else \_thesecnum \_fi}%
\_printsec{\_scantextokens{#1}}%
\_resetnonumnotoc}
\def \_insecc #1{\_par \_sectionlevel=3
\_def \_savedtitle {#1}% saved to .ref file
\_ifnonum \_else \_globaldefs=1 \_incr\_seccnum \_seccx\_fi
\_edef \_therefnum {\_ifnonum \_space \_else \_theseccnum \_fi}%
\_printsecc{\_scantextokens{#1}}%
\_resetnonumnotoc}
\public \chap \sec \secc;
```

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 L\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.

```
\def \_printrefnum #1@#2{\_leavevmode % we must be in horizontal mode
\_ifnonum \_else #1\_therefnum #2\_fi
\_ifnonum \_else \_globaldefs=1 \_incr\_chapnum \_chapx\_fi
\_edef \_therefnum {\_ifnonum \_space \_else \_thechapnum \_fi}%
\_printchap{\_scantextokens{#1}}%
\_resetnonumnotoc}
```

```
\thisoutline{⟨text⟩} saves text to the `\_theoutline` macro. `\_printrefnum` uses it and removes it.
```

```
\_def \_theoutline{}
\_def \thisoutline#1{\_gdef \_theoutline(#1)}
\_public \thisoutline 
```

```
\_def \_abovetitle{⟨penaltyA⟩}{⟨skipA⟩} and `\_belowtitle{⟨skipB⟩}` pair communicates using a special penalty 11333 in vertical mode. The `\_belowtitle` puts the vertical skip (its value is saved in `\_savedtitleskip`) followed by this special penalty. The `\_abovetitle` reads `\lastpenalty` and if it has this special value then it removes the skip used before and doesn’t use the parameter. The `\_abovetitle` creates ⟨skipA⟩ only if whatever previous skip is less or equal than ⟨skipA⟩. We must save ⟨whatever-skip⟩,
remove it, create \penaltyA (if \belowtitle does not precede) and create \whatever-skip or \skipA depending on what is greater. The amount of \skipA is measured using \setbox0=vbox.

\nbpar sets \interlinepenalty value. \nl 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 s. 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.

OpsTeX 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 \secl⟨number⟩ ⟨title-text⟩⟨eol⟩ should be used for various levels of sections (for example, when converting from Markdown to OpsTeX). \secl1 is \chap, \secl2 is \sec, \secl3 is \secc and all more levels (for ⟨number⟩> 3) are printed by the common \seclp macro. It declares only a simple design. If there is a requirement to use such more levels then the book designer can define something different here.
The `\caption/{letter}` increases \_\langle letter\rangle num counter, edefines \_\thestepnum as \_\the{letter}num and defines \_\thestep as language-dependent word using \_\mtxt, 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 a } 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. The \_\printcaption/t and \_\printcaption/f macros start in vertical mode. They switch to horizontal mode and use \_\wlabel\_\thestep as \_\label in order to make reference and hyperlink destination. They can use:

- \_\thestep expands to the word Table or Figure (depending on the current language).
- \_\thestep expands to \_\the{letter}num (caption number).

The macro \_\printcaption/t or \_\printcaption/f is processed inside group and the \_\par can be run after this group.

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:

\let\_\printcaptiona = \_\printcaptionf \let\_\everycaptiona = \_\everycaptionf
\newcount\_\anum \addto\_\secx \_\anum=0 
\sdef{\_mt:a:en}{Algorithm} \sdef{\_mt:a:cs}{Algoritmus} % + your language...

The format of the \caption text is given by the \_\captionformat/{title} macro. The default format for \_t and \_f is a paragraph in block narrower by \_\iindent and with the last line is centered. This setting is done by the \_\narrowlastlinecentered macro.
\texttt{\textbackslash eqmark} is processed in display mode (we add \texttt{\textbackslash eqno primitive}) or in internal mode when \texttt{\textbackslash aligno} is used (we don’t add \texttt{\textbackslash eqno}).

\begin{verbatim}
\optdef\_eqmark \{}\_trylabel \_ ineqmark\}
\def\_ineqmark\{\_incr\_dnum
\ifinner\else\_eqno \_fi
\_vlabel\_thednum \_hbox\{\_thednum\}%
\}
\public \eqmark ;
\end{verbatim}

The \texttt{\textbackslash numberedpar \langle letter\rangle\{\langle name\rangle\}} is implemented here.

\begin{verbatim}
\newcount\_counterA \newcount\_counterB \newcount\_counterC
\newcount\_counterD \newcount\_counterE
\def\_resetABCDE \{}\_counterA=0 \_counterB=0 \_counterC=0 \_counterD=0 \_counterE=0 \}
\def\_theAnum \{}\_othe\_chapnum.\_othe\_secnum.\_the\_counterA\}
\def\_theBnum \{}\_othe\_chapnum.\_othe\_secnum.\_the\_counterB\}
\def\_theCnum \{}\_othe\_chapnum.\_othe\_secnum.\_the\_counterC\}
\def\_theDnum \{}\_othe\_chapnum.\_othe\_secnum.\_the\_counterD\}
\def\_theEnum \{}\_othe\_chapnum.\_othe\_secnum.\_the\_counterE\}
\def\_numberedpar#1#2\{}\_ea \_incr \_csname _counter#1\_endcsname \_def\_tmpa{#1}\_def\_tmpb{#2}\_numberedparparam\}
\optdef\_numberedparparam \{}\_ea \_printnumberedpar \_csname \the\tmpa num\_ea\_endcsname \_ea\{\_tmpb\}\_numberedparparam\%
\public \_printnumberedpar ;
\end{verbatim}

The \texttt{\_printnumberedpar \_theXnum \{\langle name\rangle\}} opens numbered paragraph and prints it. The optional parameter is in \_the\_opt. You can re-define it if you need another design. \texttt{\_printnumberedpar} needs not to be re-defined if you only want to print Theorems in italic and to insert vertical skips (for example). You can do this by the following code:

\begin{verbatim}
\def\_theorem \{}\medskip\bgroup\_bf \_numberedpar A{Theorem}\}
\def\_endtheorem \{}\_par\egroup\medskip\}
\_theorem Let $M$ be... \_endtheorem
\end{verbatim}

2.27 Lists, items

\begin{verbatim}
\_codedecl \begitems \{Lists: begitems, enditems <2021-03-10}\} \%
\end{verbatim}

\texttt{\_above\_l\_skip} is used above the list of items, \texttt{\_below\_l\_skip} is used below the list of items and \texttt{\_interl\_skip} is used between items. \texttt{\_list\_skipA} is used as \texttt{\_list\_skipamount} at level 1 of items. \texttt{\_list\_skipB} is used as \texttt{\_list\_skipamount} at other levels. \texttt{\_set\_list\_skip} sets the skip dependent on the current level of items.

\begin{verbatim}
\_def \_above\_l\_skip \{}\_removelastskip \_penalty-100 \_vskip \_list\_skipamount\%
\_def \_below\_l\_skip \{}\_penalty-200 \_vskip \_list\_skipamount\%
\_def \_interl\_skip \{}\%
\_def \_list\_skipA \{}\_medskipamount\%
\_def \_list\_skipB \{}\_Opt \_plus.5 \_smallskipamount\%
\_def \_set\_list\_skip \{}\%
\_ifnum \_ilevel = 1 \_list\_skipamount = \_list\_skipA \_relax
\_else \_list\_skipamount = \_list\_skipB \_relax
\_fi\}
\end{verbatim}
The \itemnum is locally reset to zero in each group declared by \begitems. So nested lists are numbered independently. Users can set initial value of \itemnum to another value after \begitems if they want. Each level of nested lists is indented by the new \iindent from left. The default item mark is \printitem. The \begitems runs \aboveleaderskip 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).

\begin{verbatim}
42 \newcount\itemnum \itemnum=0
43 \newtoks\printitem
44 \def\begitems{\par\bgroup
45 \advance \ilevel by1
46 \setlistskip
47 \ifnum\lastpenalty<10000 \aboveleaderskip \fi
48 \itemnum=0 \adef*{\relax\ifmmode*\else ea\startitem \fi}
49 \advance \leftskip by \iindent
50 \printitem=\defaultitem
51 \the\everylist \relax
52 }\endverbatim

\def\novspaces \removelastskip \listskipamount=0pt \relax

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.\begin{verbatim}
79 \def\style#1{%
80 \ifcsname _item:#1\endcsname \printitem=\ea{\csname _item:#1\endcsname}%
81 \else \printitem=\defaultitem \fi
82 }
83 \sdef{_item:o}{\raise.4ex\hbox{$\scriptscriptstyle \bullet$} }
84 \sdef{_item:-}{- }
85 \sdef{_item:n}{\the\itemnum. }
86 \sdef{_item:N}{\the\itemnum) }
87 \sdef{_item:i}{(\romannumeral \itemnum) }
88 \sdef{_item:I}{\uppercase\ea{\romannumeral \itemnum}\kern.5em}
89 \sdef{_item:a}{\the\itemnum) }
90 \sdef{_item:A}{\the\itemnum) }
91 \sdef{_item:x}{\raise.3ex\fullrectangle{.6ex}\kern.4em}
92 \sdef{_item:X}{\raise.2ex\fullrectangle{1ex}\kern.5em}
93 \athet{\itemnum} returns the \itemnum’s lowercase letter from the alphabet.
94 \fullrectangle{\dimen} prints full rectangle with given \dimen.
\end{verbatim}
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 \TeX trick 0031 shows how to create blocks with grey background splittable to more pages.

\begin{verbatim}
def \begblock{
  \bgroup \fnset \medskip \advance \leftskip by \iindent \firstnoindent}
def \endblock{
  \par \medskip \egroup \isnextchar \par{}{\noindent}}
\end{verbatim}

\Public \begblock \endblock

2.28 Verbatim, listings

2.28.1 Inline and “display” verbatim

The internal parameters `\_ttskip`, `\_ttpenalty`, `\_viline`, `\_vifile` and `\_ttfont` for verbatim macros are set.

\begin{verbatim}
def \_ttskip{\medskip} % space above and below \begtt, \verbatim
mathchardef \_ttpenalty=100 % penalty between lines in \begtt, \verbatim
newcount \_viline % last line number in \verbatim
newread \_vifile % file given by \verbatiminput
\def \_ttfont{\tt} % default tt font
\end{verbatim}

`\code{⟨text⟩}` expands to `\detokenize{⟨text⟩}` when `\escapechar=-1`. In order to do it more robust when it is used in `\write` then it expands as noexpanded `\code{⟨space⟩}` (followed by space in its csname). This macro does the real work.

The `\_printinverbatim{⟨text⟩}` macro is used for `\code{⟨text⟩}` printing and for `’⟨text⟩’` printing. It is defined as `\_box`, so the in-verbatim `⟨text⟩` will be never broken. But you can re-define this macro.

When `\code` occurs in PDF outlines then it does the same as `\detokenize`. The macro for preparing outlines sets `\escapechar` to −1 and uses `\_regoul` token list before `\edef`.

The `\code` is not `\protected` because we want it expands to `\unexpanded{\code{⟨space⟩}{⟨text⟩}}` in `\write` parameters. This protect the expansions of the `\code` parameter (like `\`, `\` etc.).

\begin{verbatim}
def \_code#1{\unexpanded\ea{\_csname \_endcsname\#1}}
def _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 ;
\end{verbatim}

The `\_setverb` macro sets all catcodes to “verbatim mode”. It should be used only in a group, so we prepare a new catcode table with “verbatim” catcodes and we define it as `\catcodetable \verbatimcatcodes`. After the group is finished then original catcode table is restored.

\begin{verbatim}
def \_newcatcodetable \verbatimcatcodes
\def \_setverb\{\beginninggroup
  \def _dodef{\catcode’\#1=12 }
  \dospecials
  \savelist{\verbatimcatcodes}{% all characters are normal
    \endgroup}
  \setverb
}
\end{verbatim}

`\verbchar{⟨char⟩}` saves original catcode of previously declared `⟨char⟩` (if such character was declared) using `\_savedttchar` and `\_savedttcharc` values. Then new such values are stored. The declared character is activated by `\_adef` as a macro (active character) which opens a group, does `\_setverb` and other settings and reads its parameter until second the same character. This is done by the `\_readverb` macro. Finally, it prints scanned `⟨text⟩` by `\_printinverbatim` and closes group. Suppose that `\verbchar“` is used. Then the following work is schematically done:
\def "{\begingroup \setverb ... \readverb}
\def \readverb #1"{\printinverbatim{#1}\endgroup}

Note that the second occurrence of \" is not active because \setverb deactivates it.

\begtt is defined only as public. We don’t need a private \begtt variant. This macro opens a group and sets % as an active character (temporary). This will allow it to be used as the comment character at the same line after \begtt. Then \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 \~ and reads the following text in verbatim mode until \endtt occurs. This scanning is done by \startverb macro which is defined as:

\def \startverb #1\endtt #2\~{}{...}

We must to ensure that the backslash in \endtt has category 12 (this is a reason of the \\ chain in real code). The #2 is something between \endtt and the end of the same line and it is simply ignored.

The \startverb puts the scanned data to \prepareverbdata. It sets the data to \tmpb without changes by default, but you should re-define it in order to do special changes if you want. (For example, \hisyntax redefines this macro.) The scanned data have \~ at each end of line and all spaces are active characters (defined as \␣). Other characters have normal category 11 or 12.

The \~ 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 \~ 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.

136
The \_printverbline\{\langle line\rangle\} expects that it starts in vertical mode and it must do \par to return the vertical mode. The \_printverblinenum is used here: it does nothing when \ttline<0 else it prints the line number using \llap.

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

The \ttline is increased here in the \_printverb macro because of comments-blocks: the \_prinverbline 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 \vscanparameter and \vscanminus. Finally the \doverbinput is run. At the beginning of \doverbinput, we have \viline = number of lines already read using previous \verbinput, \vinolines = the number of lines we need to skip and \vidolines = the number of lines we need to print. A similar preparation is done as in \begtt after the group is opened. Then we skip \vinolines lines in a loop a and we read \vidolines lines. The read data is accumulated into \tmpb macro. The next steps are equal to the steps done in \startverb macro: data are processed via \prepareverbdata and printed via \printverb loop.
\_def\_viscanminus(#1-#2)%
\_if$#1$\_tmpnum=0
\_else \_tmpnum=#1 \_advance\_tmpnum by-1 \_fi
\_ifnum\_tmpnum<0 \_tmpnum=0 \_fi % (0-13) = (1-13)
\_edef\_vinolines{\_the\_tmpnum}%
\_if$#2$\_tmpnum=0
\_else \_tmpnum=#2 \_advance\_tmpnum by-\_vinolines \_fi
\_edef\_vidolines{\_the\_tmpnum}%
\_doverbinput
}
\_def\_doverbinput{%
\_tmpnum=\_vinolines
\_advance\_tmpnum by-\_viline
\_openin\_vifile{\_vifilename}%
\_global\_viline=0
\_else
\_edef\_vinolines{\_the\_tmpnum}%
\_fi
\_vskip\_parskip \_ttskip \_wipeepar \_setxhsize
\_begingroup
\_ifnum\_ttline<-1 \_let\_printverblinenum=\_relax \_else \_initverblinenum \_fi
\_setverb \_adef{ }{\_dsp}\_adef\^^I{\t}\_parindent=\_ttindent \_parskip=0pt
\_def\t{\_hskip \_dimexpr\_tabspaces em/2\_relax}%
\_protrudechars=0 % disable protrusion
\_the\_everytt_=10 \_catcode\%=9 % used in \commentchars comments
\_ea\_printverb \_tmpb\_fin
\_ttlinesave
\_par \_restoremathsb
\_endgroup
\_ttskip
\_isnextchar\par{}{\_noindent}\_fi
\_endfor
}
\_def\_vireadline{\_read\_vifile to \_tmp \_incr\_viline }
\_def\_visaveline{\_ea\_addto\_ea\_tmpb\_ea{\_tmp}}

\_public \verbinput ;
\_savemathsb, \_restoremathsb pair is used in \begtt...\endtt or in \verbinput to temporary suppress the \mathsbon because we don’t need to print \int _a in verbatim mode if \int _a is really written. The \_restoremathsb is defined locally as \mathsbon only if it is needed. \verbinput (4-) optex.lua here and the \_commentchars -- was set before it.

If the language of your code printed by \verbinput supports the format of comments started by two characters from the beginning of the line then you can set these characters by \_commentchars(first)(second). Such comments are printed in the non-verbatim mode without these two characters and they look like the verbatim printing is interrupted at the places where such comments are. See the section 2.39 for good illustration. The file optex.lua is read by a single command \verbinput (4-) optex.lua here and the \_commentchars -- was set before it.
If you need to set a special character by \commentchars then you must to set the catcode to 12 (and space to 13). Examples:

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

There is one limitation when TEX interprets the comments declared by \commentchars. Each block of comments is accumulated to one line and then is re-interpreted by 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.

\verbatim.opm
350 \_def\_vcomments{}
351 \_let\_hicomments=\_relax
352
353 \_def\_commentchars#1#2{\_def\_testcommentchars ##1##2##3\_relax ##4\_iftrue{\_ifnum % not closed in this macro
354 \_ifx #1##1\_ifx#2##21\_fi\_fi 0>0
355 \_ifx\_relax##3\_relax \_addto\_vcomments{\_endgraf}% empty comment=\enfgraf
356 \else \_addto\_vcomments{##3}\_fi\%}
357 \_def\_hicomments{\_replfromto{\b\n#1#2}{^^J}{\w{#1#2####1}^^J}}% used in \_hissyntax
358 \_def\_testcommentchars #1\_iftrue{\_iffalse} % default value of \_testcommentchar
359 \_def\_printcomments{\_ttskip
360 \_catcodetable0 \_rm \_everypar={}\_par}
361 \_ttskip
362 {\_catcodetable0 \_rm \everypar={}}
363 \_noindent \_ignorespaces \_scantextokens\_ea{\_vcomments}\_par\%}
364 \_ttskip
365 }
366 \_public \_commentchars ;

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.

\verbatim.opm
377 \_def \_visiblesp{\_ifx\_initunifonts\_relax \_def\_dsp{\_char9251 }%
378 \else \_def\_dsp{\_char32 }\_fi}
379 \_let\_dsp=\% primitive "direct space"
380 \_ttskip
381 \_public \_visiblesp ;

2.28.2 Listings with syntax highlighting

The user can write

\begtt \_hisyntax{C}
...
\endtt

to colorize the code using C syntax. The user can also write \everytt={\_hisyntax{C}} to have all verbatim listings colorized.
\hisyntax{⟨name⟩} reads the file hisyntax-{⟨name⟩}.opm where the colorization is declared. The parameter ⟨name⟩ is case insensitive and the file name must include it in lowercase letters. For example, the file hisyntax-c.opm looks like this:

```latex
\_codedecl \hisyntaxc (Syntax highlighting for C sources <2020-04-03>)
\_newtoks \hisyntaxc \_newtoks \hicolors
global\_hicolors={% colors for C language
    \hicolor K \Red % Keywords
    \hicolor S \Magenta % Strings
    \hicolor C \Green % Comments
    \hicolor N \Cyan % Numbers
    \hicolor P \Blue % Preprocessor
    \hicolor O \Blue % Non-letters
}
global\_hisyntaxc={%}
    \the\_hicolors
    let\c=\_relax let\e=\_relax let\o=\_relax
    replfromto {/*}{*/} {{\x C{/*/1*/}}} % /*...*/
    replfromto {//}{^^J} {{\x C{//#1}^^J}} % //...
    replthis {\_string#}{^^J} {{\x P{##1}^^J}} % #include ...
    replfromto {\_string}"}{^^J} {{\x S{"#1}^^J}} % " protected inside strings
    replthis {(^)} {{\z K{#1}}}
    replfromto {.}{\z .}
    replthis {e\#1e\#2}{e+} replthis {e\#1e\#2}{e-}
    replthis {E\#1E\#2} replthis {E\#1E\#2} replthis {E\#1E\#2} replthis {E\#1E\#2}
    def\#1{\x O{#1}}
    def\#1{\x N{#1}}
}
```

OpTEX provides hisyntax-{c,python,tex,html}.opm files. You can take inspiration from these files and declare more languages.

Users can re-declare default colors by \hicolors={⟨list of color declarations⟩}. This value has precedence over \hicolors⟨name⟩ values declared in the hicolors-{⟨name⟩}.opm file. For example \hicolors={\hicolor S \Brown} causes all strings in brown color.

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

Notes for hi-syntax macro writers
The file hisyntax-{⟨name⟩} .opm is read only once and in a TEX group. If there are definitions then they must be declared as global.

The file hisyntax-{⟨name⟩} .opm must (globally) declare \hisyntax{⟨name⟩} token list where the action over verbatim text is declared typically by using the \replfromto or \replthis macros.

The verbatim text is prepared by the pre-processing phase, then \hisyntax{⟨name⟩} is applied and then the post-processing phase does final corrections. Finally, the verbatim text is printed line by line.

The pre-processing phase does:

- Each space is replaced by \n\n\n, so \n⟨word⟩\n is the pattern for matching whole words (no subwords). The \n control sequence is removed in the post-processing phase.

OpTEX provides hisyntax-{c,python,tex,html}.opm files. You can take inspiration from these files and declare more languages.

Users can re-declare default colors by \hicolors={⟨list of color declarations⟩}. This value has precedence over \hicolors⟨name⟩ values declared in the hicolors-{⟨name⟩}.opm file. For example \hicolors={\hicolor S \Brown} causes all strings in brown color.

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

Notes for hi-syntax macro writers
The file hisyntax-{⟨name⟩} .opm is read only once and in a TEX group. If there are definitions then they must be declared as global.

The file hisyntax-{⟨name⟩} .opm must (globally) declare \hisyntax{⟨name⟩} token list where the action over verbatim text is declared typically by using the \replfromto or \replthis macros.

The verbatim text is prepared by the pre-processing phase, then \hisyntax{⟨name⟩} is applied and then the post-processing phase does final corrections. Finally, the verbatim text is printed line by line.

The pre-processing phase does:

- Each space is replaced by \n\n\n, so \n⟨word⟩\n is the pattern for matching whole words (no subwords). The \n control sequence is removed in the post-processing phase.
• Each end of line is represented by \n--J\n.
• The \_start control sequence is added before the verbatim text and the \_end control sequence is appended to the end of the verbatim text. Both are removed in the post-processing phase.

Special macros are working only in a group when processing the verbatim text.
• \n represents nothing but it should be used as a boundary of words as mentioned above.
• \t represents a tabulator. It is prepared as \n\t\n because it can be at the boundary word boundary.
• \x \{letter\} \{text\} can be used as replacing text. Consider the example

\replfromto{/\{*\}}{/\*\{x C{/#1*/}}

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

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

\x C{\{text1\}--J\{text2\}--J\{text3\}}

is replaced by

\z C{\{text1\}--J\z C{\{text2\}--J\z C{\{text3\}}}

\z \{letter\} \{text\} is expanded to \z: \{letter\} \{text\} and if \hicolor \{letter\} \{color\} is declared then \z: \{letter\} \{text\} expands to \{\color\} \{text\}. So, required color is activated for each line separately (e.g. for C comments spanning multiple lines).
• \y \{text\} is replaced by \y \{text\} in the post-processing phase. It should be used for macros without a parameters. You cannot use unprotected macros as replacement text before the post-processing phase, because the post-processing phase is based on the expansion of the whole verbatim text.

The macros \replfromto and \replthis manipulate the verbatim text that is already stored in the \_tmpb macro.
\replfromto \{from\} \{to\} \{replacement\} finds the first occurrence of \{from\} and the first occurrence of \{to\} following it. The \{text\} between them is packed into \#1 and available to \{replacement\} which ultimately replaces \{text\}.

\replfromto continues by finding next \{from\}, then, next \{to\} repeatedly over the whole verbatim text. If the verbatim text ends with opening \{from\} but has no closing \{to\}, then \{to\} is appended to the verbatim text and the last part of the verbatim text is replaced too.

The first two parameters are expanded before use of \replfromto. You can use \csstring\% or something else here.

The \replthis \{pattern\} \{replacement\} replaces each \{pattern\} by \{replacement\}. Both parameters of \replthis are expanded first.

The patterns \{from\}, \{to\} and \{pattern\} are not found when they are hidden in braces \{\ldots\}. E.g.
\replfromto{/**}*/{\x C{/**#1/**}} replaces all C comments by \x C{\ldots}. The patterns inside \ldots are not used by next usage of \replfromto or \replthis macros.

The \_xscan macro replaces occurrences of \x by \z in the post-processing phase. The construct \x \langle letter \rangle \{ \langle text \rangle \} expands to \_xscan \{ \langle letter \rangle \} \{ \langle text \rangle \}'''. \z. If #3 is \_fin then it signals that something wrong happens, the \langle from \rangle was not terminated by legal \langle to \rangle when \replfromto did work. We must to fix this by using the \_xscanR macro.

\_def\_xscan#1#2\_fin#3{\_ifx\_fin#3 \_ea\_xscanR \_fi
\z{#1}{#2}%
\_ifx^#3\_else \_afterfi{\_xscan{#1}#3}\_fi}

\_def\_xscanR#1\_fin#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}}}

\hisyntax{\langle name \rangle} re-defines default \_prepareverbdata \langle macro \rangle \langle verbtext \rangle, but in order to do it does more things: It saves \langle verbtext \rangle to \_tmpb, appends \n around spaces and ''\n characters in pre-processing phase, opens \hisyntax – \langle name \rangle.opm file if \_hisyntax \langle name \rangle is not defined. Then \_the \_hisyntax \langle name \rangle is processed. Finally, the post-processing phase is realized by setting appropriate values to the \x and \y macros and doing \_edef \_tmpb{\_tmpb}.

\hi-syntax{\langle name \rangle} aliases for languages can be declared like this. When \hisyntax{xml} is used then this is the same as \hisyntax{html}.

\_sdef{\_hialias:xml}{html}
\_sdef{\_hialias:json}{c}

2.29 Graphics

The \inspic is defined by \pdfximage and \pdfrefximage primitives. If you want to use one picture more than once in your document, then the following code is recommended:
My picture: \copy\mypic, again my picture: \copy\mypic, etc.

This code downloads the picture data to the PDF 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 \transformbox{\{transformation\}\{\copy\mypic\}}.

\inspic accepts old syntax \inspic {\texttt{filename}}{\texttt{space}} or new syntax \inspic {\texttt{filename}}. So, we need to define two auxiliary macros \_inspicA and \_inspicB.

You can include more \pdfximage parameters (like page{\texttt{number}}) in the \_picparams macro.

All \inspic macros are surrounded in \hbox in order user can write \moveright\inspic {\texttt{...}} or something similar.

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.
The \transformbox\{\text{\textit{transformation}}}\{\text{\textit{text}}} is copied from OPmac trick 0046. 

The \rotbox\{\text{\textit{degrees}}}\{\text{\textit{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.
The ratio \incircle{<text>} puts the \text<text> to desired place: From current point moves \langle down and \langle right>, puts the \langle text> and returns back. The current point is unchanged after this macro ends.

The \putpic{<right> \langle up} \langle text > puts the \langle text > to desired place: From current point moves \langle down and \langle right>, puts the \langle text> and returns back. The current point is unchanged after this macro ends.

The incircle{<image-file>} puts the image to the background of each page. It is used in the \slides style, for example.

\input{example}
it means filling color.

The \_setflcolors 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.

\_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}}
\_optdef\_inoval[\_]{\_vbox\_bgroup}
\_roundness=2pt \_fcolor=\_Yellow \_lcolor=\_Red \_lwidth=.5bp
\_shadow=N \_overlapmargins=N \_bkern=Opt \_vkern=Opt
\_the\_ovalparams \_relax \_the\_opt \_relax
\_touppervalue\_overlapmarginsvalue \_touppervalue\_shadowvalue
\_ifs\_overlapmarginsvalue \_dimen0=\_roundness \_dimen1=\_roundness
\_else \_dimen0=\_bkern \_dimen1=\_vkern \fi
\_setbox0=\_hbox\_bgroup\_bgroup \_aftergroup\_inovalA \_kern\_bkern \_let\_next=\%
\_optdef\_incircle[\_]{\_vbox\_bgroup}
\_ratio=1 \_fcolor=\_Yellow \_lcolor=\_Red \_lwidth=.5bp
\_shadow=N \_overlapmargins=N \_bkern=3pt \_vkern=3pt
\_ea\_the \_ea\_circleparams \_space \_relax
\_ea\_the \_ea\_opt \_space \_relax
\_touppervalue\_overlapmarginsvalue \_touppervalue\_shadowvalue
\_setbox0=\_hbox\_bgroup\_bgroup \_aftergroup\_incircleA \_kern\_bkern \_let\_next=\%
\_optdef\_incircleA[\_]{\_vbox\_bgroup}
\_ratio=1 \_fcolor=\_Yellow \_lcolor=\_Red \_lwidth=.5bp
\_shadow=N \_overlapmargins=N \_bkern=3pt \_vkern=3pt
\_ea\_the \_ea\_circleparams \_space \_relax
\_ea\_the \_ea\_opt \_space \_relax
\_touppervalue\_overlapmarginsvalue \_touppervalue\_shadowvalue
\_setbox0=\_hbox\_bgroup\_bgroup \_aftergroup\_incircleA \_kern\_bkern \_let\_next=\%
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 \addpagemanager{(resource type)}{(PDF name)}{(PDF dictionary)}. You can use \pageresources or \pagemanager if you need to insert resource entries to manually created PDF XObjects.

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 \shadow{(curve)} does the shadow effect.
A generic macro \(\text{\_clipinpath}\langle x \rangle \langle y \rangle \langle \text{curve} \rangle \langle \text{text} \rangle\) declares a clipping path by the \(\langle \text{curve} \rangle\) 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 \(\text{bp}\) without the unit here. The macros \(\text{\_clipinoval}\langle x \rangle \langle y \rangle \langle \text{width} \rangle \langle \text{height} \rangle \{\langle \text{text} \rangle\}\) and \(\text{\_clipincircle}\langle x \rangle \langle y \rangle \langle \text{width} \rangle \langle \text{height} \rangle \{\langle \text{text} \rangle\}\) are defined here. These macros read normal \TeX\ dimensions in their parameters.

### 2.30 The \texttt{table} macro, tables and rules

#### 2.30.1 The boundary declarator

The \(\langle \text{declaration} \rangle\) part of \texttt{table}\{\(\langle \text{declaration} \rangle\}\}{\(\langle \text{data} \rangle\}\) includes column declarators (letters) and other material: the \(\mid\) or \(\langle \text{cmd} \rangle\). 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 \{c|c\(\times\times\)(yy)c\} should be written more exactly using the boundary declarator : by \{c|c\(\times\times\)(yy)c\}. But you can set these boundaries to other places using the boundary declarator : explicitly, for example \{c|c\(\times\times\)(yy)c\}. The boundary declarator : can be used only once between each pair of column declarators.

Each table item has its group. The \(\langle \text{cmd} \rangle\) 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 \(\langle \text{setting} \rangle\) followed by column declarator. But if the column is not first, you must use \(\langle \text{setting} \rangle\). Example. We have three centered columns, the second one have to be in bold font and the third one have to be in red:

\[
\text{table}\{c:\text{\bf}\langle cxx\rangle\langle yy\rangle}\text{c}\{\langle \text{data} \rangle\}\]

#### 2.30.2 Usage of the \texttt{\tabskip} primitive

The value of \texttt{\tabskip} is used between all columns of the table. It is glue-type, so it can be stretchable or shrinkable, see next section 2.30.3. By default, \texttt{\tabskip} is 0pt. It means that only \texttt{\tabiteml}, \texttt{\tabitemr} and \(\langle \text{cmd} \rangle\) 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 `\everytable` 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 `\tabskip=⟨value⟩` in the `⟨declaration⟩` immediately before boundary character. The boundary character represents the column pair for which the `\tabskip` has individual value. For example `c(\tabskip=5pt):r` gives `\tabskip` value between `c` and `r` columns. You need not use boundary character explicitly, so `c(\tabskip=5pt)r` 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⟨size⟩{⟨declaration⟩}{⟨data⟩}` uses stretchability or shrinkability of all spaces between columns generated by `\tabskip` value and eventually by `\tabskipl`, `\tabskipr` values. See example below.
- `\table pxto⟨size⟩{⟨declaration⟩}{⟨data⟩}` expands the columns declared by `p{⟨size⟩}`, if the `⟨size⟩` is given by a virtual `\tsize` unit. See the example below.

Example of \table to⟨size⟩:

```
\hst\table{\tabskip=0pt plus1fil minus1fil}\
\table to\hsize {lr}{⟨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 \table pxto⟨size⟩ (means “paragraphs expanded to”):

```
\table pxto\hsize {l|c|p{\tsize}|}{\crl
  aaa & Ddkas jd dsjd s ds cgha sfgs dd fddzf dfhz xxz
  dras ffg hksd kds d sdjd s 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 `c` column is variable width (it gets the width of the most wide item) and the resting space to given \hsize is filled by the `p` column.

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

```
\table pxto13cm {r p{3.5\tsize} p{2\tsize} p{\tsize} l}{⟨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⟩}` behaves like `\hbox{⟨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 ⟨dim:label⟩{\hss ⟨text⟩\hss}`, where ⟨dim: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 \{\langle label \rangle\{\langle dimen \rangle \text{ expands to } \langle dim:label \rangle \text{ if this value is known, else it expands to the given } \langle dimen \rangle.

The optional parameter r or l can be written before \{\langle label \rangle \} \{\langle text \rangle \} if you want to put the text to the right or to the left side of the box width.

Try the following example and watch what happens after first \TeX run and after the second one.

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

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 \rangle. The \_ifpxto returns true in this case.

The \table continues by reading \{\langle declaration \rangle \} in the \_tableA macro. Catcodes (for example the | character) have to be normal when reading \table parameters. This is the reason why we use \catcodetable here.

The \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.

The \table\{\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 \replstring to prefix each macro \_crl (etc.) by \_cocr. See \_tabreplstrings. The whole \_tableB macro is hidden in {...} 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\}\crcr}
This sets the desired boundary values of \tabskip. The “between-columns” values are set as \tabskip=\_tabskipmid in the \langle converted declaration\rangle 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{.\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 in the second pass.

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.

\_tableb #1{\_egroup
\_def\_tmpb{#1}\_tablereplstrings
\_edef\_tabskipmid{\_the\_tabskip}\_tabskip=\_tabskipl
\_ifpxto
\_edef\_tsizes{\_global\_tsizesum=\_the\_tsizesum \_gdef\_noexpand\_tsizelast{\_tsizelast}}%
\_tsizesum=\zo \_def\_tsizelast{0}%
\_tsize=-\hsize \_setbox0=\vbox{\_tablepxpreset \_halign \_tableC}%
\_advance\_tmpdim by-\_wd0
\_ifdim \_tmpdim >\zo \_else \_tsizesum=\zo \_fi
\_ifdim \_tsizesum >\zo \_tsize =\expr{\_number\hsize/\_number\_tsizesum}\_tmpdim
\_else \_tsize=\zo \_fi
\_tsize % retoring values if there is a \table pxto inside a \table pxto.
\_setbox0=\null \_halign \_tableC
\_else
\_halign \_tablew \_tableC
\_fi
\_egroup % \_tablebox\_bgroup is in the \_table macro
}

\_def\_tableC{\_ea{\_the\_tabdata\_tabskip=\_tabskipr\_cr \_scantextokens\_ea{\_tmpb\_crcr}}}
\_tablereplstrings replaces each \crl etc. to \crcr\crl. The reason is: we want to use macros that scan its parameter to a delimiter written in the right part of the table item declaration. The \crcr cannot be hidden in another macro in this case.

\long\_def\_tablereplstrings{\_replstring\_tmpb{\crl}{\_crcr\crl}\_replstring\_tmpb{\crll}{\_crcr\crll}\_replstring\_tmpb{\crli}{\_crcr\crli}\_replstring\_tmpb{\crlli}{\_crcr\crlli}\_replstring\_tmpb{\crlp}{\_crcr\crlp}}

\_def\_tablepxpreset{} % can be used to de-activate references to .ref file
\_newbox\_tstrutbox % strut used in table rows
\_newtoks\_tabdata % the \halign declaration line

The \_scantabdata macro converts \table’s \langle declaration\rangle to \halign \langle converted declaration\rangle. The result is stored into \_tabdata tokens list. For example, the following result is generated when \langle declaration\rangle={\{\| \cr \]].
\tabdata: \_vrule\_the\_\_tabiteml{\_hfil\#\_unsskip\_hfil}\_the\_\_tabitemr\_tabstrutA &\_the\_\_tabiteml{\_hfil\#\_unsskip}\_the\_\_tabitemm
&\_the\_\_tabiteml{\_hfil\#\_unsskip\_hfil}\_the\_\_tabitemr
&\_the\_\_tabiteml{\_relax\#\_unsskip\_hfil}\_the\_\_tabitemm\_vrule\_\_tabstrutA
&\_the\_\_tabiteml{\_hfil\#\_unsskip\_hfil}\_the\_\_tabitemr\_tabstrutA
&\_the\_\_tabiteml{\_relax\#\_unsskip\_hfil}\_the\_\_tabitemm\_vrule\_\_tabstrutA
\ddlinedata: &\_dditem &\_dditem\_vvitem &\_dditem &\_dditem

The second result in the \_ddlinedata macro is a template of one row of the table used by \crli macro.
The \_addtabitemx adds the boundary code (used between columns) to the \textit{(converted declaration)}. This code is \texttt{\egroup &\bgroup \colnum=(value)\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 \texttt{\langle data\rangle} item because \texttt{\halign} expands the front of the item and the left part of the declaration is processed after this. Use \texttt{\relax\the\colnum} instead. Or you can write:

\texttt{\def\showcolnum{\ea\def\ea\totcolnum\ea{\the\colnum}/\totcolnum}}

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 \texttt{\table \langle declaration\rangle} to the \textit{(converted declaration)}.

The default “declaration letters” c, l, r and p are declared by setting \texttt{\_tabdeclarec, \_tabdeclarcl, \_tabdeclarer} and \texttt{\_paramtabdeclarerp} macros. In general, define \texttt{\def\_tabdeclare\{letter\}{...}} for a non-parametric letter and \texttt{\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:

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

We don’t want vertical line after red A to be in red.
The \_paramtabdeclarep\{\textit{data}\} is invoked when p\{\textit{data}\} declarator is used. First, it saves the \texttt{\hsize} value and then it runs \_tablepar. The \_tablepar macro behaves like \_tableparbox (which is \texttt{\vtop}) in normal cases. But there is a special case: if the first pass of \texttt{pxto table} is processed then \texttt{\hsize} is negative. We print nothing in this case, i.e. \_tableparbox is \texttt{\ignoreit} and we advance the \texttt{\_tsizesum}. The auxiliary macro \_tsizelast is used to do advancing only in the first row of the table. \_tsizesum and \_tsizelast are initialized in the \_tableB macro.

The \_tableparB initializes the paragraphs inside the table item and \_tablepar closes them. They are used in the \_paramtabdeclarep macro. The first paragraph is no indented.

Users put optional spaces around the table item typically, i.e. they write \&text\& instead of &text\&. The left space is ignored by the internal \TeX{} algorithm but the right space must be removed by macros. This is a reason why we recommend to use \_unsskip after each ## in your definition of “declaration letters”. This macro isn’t only the primitive \texttt{\unskip} because we allow usage of plain \TeX{} \texttt{\hideskip} macro: \&\hideskip text\hideskip&.

The \_unsskip macro is more tricky. The \_tableparbox isn’t printed immediately, but \texttt{\setbox2=\unvcopy\_unskip\_global\_setbox1=} is prefixed by the macro \_tableparA, which is empty by default (used in \_tablepar). The \_tablepar is processed after the box is set: it checks if there is only one line and prints \texttt{\hbox to\texttt{\hsize}\{\hfil\langle this line\rangle\hfil\}} in this case. In other cases, the box is printed.
The family of \_cr macros \_cr, \_crll, \_crli, \_crlli, \_crlp and \_tskip \langle dimen \rangle is implemented here. The \_zerotabrule is used to suppress the negative \lineskip declared by \tablinespace.

\begin{verbatim}
  \_def\_crl{\_ncrl\_noalign{\_hrule}}
  \_def\_crll{\_ncrl\_noalign{\_hrule\_kern\_hhkern\_hrule}}
  \_def\_zerotabrule {\_noalign{\_hrule height\_zo width\_zo depth\_zo}}
  \_def\_crli{\_crl\_zerotabrule\_omit}
  \_gdef\_dditem{\_omit\_tablinefil}\_gdef\_vvitem{\_kern\_vvkern\_vrule}\_gdef\_vvitemA{\_vrule}\_vvleft\_tablinefil\_ddlinedata\_crcr\_zerotabrule
  \_def\_crlli{\_crli\_noalign{\_kern\_hhkern}\_crli}
  \_def\_tablinefil{\_leaders\_hrule\_hfil}
  \_def\_crlp#1{\_crl\_zerotabrule\_noalign{\_kern-\_drulewidth}\_omit\_xdef\_crlplist{#1}\_xdef\_crlplist{,\_ea}\_ea\_crlpA\_crlplist,\_fin,\_global\_tmpnum=0\_gdef\_dditem{\_omit\_crlpD}\_gdef\_vvitem{\_kern\_vvkern\_kern\_drulewidth}\_gdef\_vvitemA{\_kern\_drulewidth}\_vvleft\_crlpD\_ddlinedata\_global\_tmpnum=0\_crcr\_zerotabrule}
  \_def\_crlpA#1,{\_ifx\_fin#1\_else\_crlpB#1-\_fin,\_ea\_crlpA\_fi}
  \_def\_crlpB#1#2-#3,{\_ifx\_fin#3\_xdef\_crlplist{\_crlplist#1#2,}\_else\_crlpC#1#2-#3,\_fi}
  \_def\_crlpC#1-#2-#3,{\_tmpnum=#1\_relax\_loop\_xdef\_crlplist{\_crlplist\_the\_tmpnum,}\_ifnum\_tmpnum<#2\_advance\_tmpnum by1\_repeat}
  \_def\_crlpD{\_incr\_tmpnum\_edef\_tmpa{\_noexpand\_isinlist\_noexpand\_crlplist{,\_the\_tmpnum,}}\_tmpa\_iftrue\_kern-\_drulewidth\_tablinefil\_kern-\_drulewidth\_else\_hfil\_fi}
  \_def\_mspan{\_omit\_afterassignment\_mspanA\_mscount=}\_def\_mspanA[#1]#2{\_loop\_ifnum\_mscount>1\_cs{\_span}\_omit\_advance\_mscount-1\_repeat\_count1=\_colnum\_colnum=0\_def\_tmpa{}\_tabdata={}\_scantabdata#1\_relax\_colnum=\_count1\_setbox0=\_vbox{\_halign\_ea{\_the\_tabdata\_cr#2\_cr}\_global\_setbox8=\_lastbox}{\_setbox0=\_hbox{\_unhbox8\_unskip\_global\_setbox8=\_lastbox}\_unhbox8\_ignorespaces}\_public \_mspan ;}
  \_def\_vspan#1#2#{\_vspanA{#1#2}}\_def\_vspanA#1#2{\_vtop to\_zo{\_hbox{\_lower\_dimexpr#1\_dimexpr(\_ht\_tstrutbox+\_dp\_tstrutbox)/2\relax\_ht\_tstrutbox+\_dp\_tstrutbox)/2\relax\_hbox{#2}}\_vss}}\_public \_vspan ;
\end{verbatim}

The \_mspan{\langle number \rangle}{\langle declaration \rangle}{\langle text \rangle} macro generates similar \omit\span\omit\span sequence as plain \TeX macro \multispan. Moreover, it uses \_scantabdata to convert \langle declaration \rangle from \table syntax to \halign syntax.

\begin{verbatim}
  \_def\_vspan#1#2#{\_vspanA{#1#2}}\_def\_vspanA#1#2{\_vtop to\_zo{\_hbox{\_lower\_dimexpr#1\_dimexpr(\_ht\_tstrutbox+\_dp\_tstrutbox)/2\relax\_ht\_tstrutbox+\_dp\_tstrutbox)/2\relax\_hbox{#2}}\_vss}}\_public \_vspan ;
\end{verbatim}

The \_vspan{\langle number \rangle}{\langle text \rangle} implementation is here. We need to lower the box by \langle number \rangle-1*/(ht+dp) of \tabstrut. The \_mspan{\langle number \rangle}{\langle declaration \rangle}{\langle text \rangle} macro generates similar \omit\span\omit\span sequence as plain \TeX macro \multispan. Moreover, it uses \_scantabdata to convert \langle declaration \rangle from \table syntax to \halign syntax.

The \_vspan{\langle number \rangle}{\langle text \rangle} implementation is here. We need to lower the box by \langle number \rangle-1*/(ht+dp) of \tabstrut. The \_mspan{\langle number \rangle}{\langle declaration \rangle}{\langle text \rangle} macro generates similar \omit\span\omit\span sequence as plain \TeX macro \multispan. Moreover, it uses \_scantabdata to convert \langle declaration \rangle from \table syntax to \halign syntax.
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 \hhkern parameters.}

\texttt{\vbox and \eqboxsize are implemented here. The widths of all \eqboxes 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 \texttt{\_eqb:\langle label\rangle}.

\texttt{\betweencolumns or \leftofcolumns or \rightofcolumns include a material printed between columns or left of all columns or right of all columns respectively. The \betweencolumns must include a stretchability or a material with exactly \colsep width. You can redefine these macros. For example the rule between columns can be reached by \texttt{\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, \url{http://petr.olsak.net/tbn.html}, but in Czech. Roughly speaking, macros complete all material between \begmulti\langle num-columns\rangle and \endmulti into one \vbox. Then the macro measures the amount of free space at the current page using \texttt{\pagegoal and \pagtotal and does \vspl 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 \hbox{\langle columns\rangle} 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 \texttt{\widowpenalty and \clubpenalty} (they are set to 10000 in Op\TeX).}
\_divide\hspace by\_Ncols \_advance\hspace by\_\colsep
\_mullines=0
\_def\_par(\_if\_hmode\_endgraf\_global\_advance\_mullines by\_prevgraf\_fi)%
}\_def\_endmulti\{\_vskip\_\prevdepth\_vf1l
\_ea\_egroup\_ea\_egroup\_ea\_baseline\_skip\_the\_ baseline\_skip\_relax
\_dimen0=.8\_maxdimen \_tmpnum=\_dimen0 \_divide\_tmpnum by\_baselineskip
\_splittopskip\_baselineskip
\_setbox1=\_vsplit6 to0pt % initialize first \splittopskip in \box6
%% \dimen1 := the free space on the page
\_penalty0 % initialize \_pageoal
\_ifdim\_pagegoal=\_maxdimen \_setcolsize\_vsize
\_else \_setcolsize(\_dimexpr\_pagegoal-\_\pagetotal)\_fi
\_ifdim \_dimen1<2\_baselineskip
\_vfil\_break \_setcolsize\_vsize \_fi
\_ifnum\_mullines<\_tmpnum \_dimen0=\_ht6 \_else \_dimen0=.8\_maxdimen \_fi
\_divide\_dimen0 by\_Ncols \_relax
%% split the material to more pages?
\_ifvoid6 \_else
\_ifdim \_dimen0>\_dimen1 \_splitpart
\_else \_balancecolumns \_fi \_only balancing
\_multiskip \_egroup
\}

Splitting columns...
\_def\_makecolumns\{\_bgroup % full page, destination height: \dimen1
\_vbadsness=20000 \_dimen6=\_wd6
\_createcolumns
\_printcolumns
\_dimen0=\_dimen1 \_divide\_dimen0 by\_baselineskip \_multiply\_dimen0 by\_Ncols
\_global\_advance\_mullines by\_\_dimen0
\_egroup
\}
\_def\_splitpart\{%
\_makecolumns \_full page
\_vskip 0pt plus 1fil minus\_baselineskip \_break
\_ifnum\_mullines=\_ht6 \_else \_dimen0=\_ht6 \_dimen0=.8\_maxdimen \_fi
\_divide\_dimen0 by\_Ncols \_relax
\_if\_balancecolumns\_flushcolumns \_advance\_dimen0 by-.5\_vsize \_fi
\_setcolsize\_vsize \_dimen2=\_dimen1
\_advance\_dimen2 by-.5\_baselineskip
%% split the material to more pages?
\_ifvoid6 \_else
\_ifdim \_dimen1>\_dimen0 \_ea\_ea\_ea \_splitpart
\_else \_balancecolumns \_fi \_last balancing
\_fi \_fi
\}

Final balancing of the columns.
\_def\_balancecolumns\{\_bgroup \_setbox7=\_copy6 % destination height: \dimen0
\_ifdim\_dimen0>\_baselineskip \_break
\_ifnum\_mullines=\_ht6 \_else \_dimen0=\_ht6 \_dimen0=.8\_maxdimen \_fi
\_divide\_dimen0 by\_Ncols \_relax
\_if\_balancecolumns\_flushcolumns \_advance\_dimen0 by-.5\_vsize \_fi
\_setcolsize\_vsize \_dimen2=\_dimen1
\_advance\_dimen2 by-.5\_baselineskip
%% split the material to more pages?
\_ifvoid6 \_else
\_ifdim \_dimen1>\_dimen0 \_ea\_ea\_ea \_splitpart
\_else \_balancecolumns \_fi \_last balancing
\_fi \_fi
\}
\_setcolsize(\_dime)\_sets initial value \_dime=(\_size) which is used as height of columns at given page. The correction \splittopskip—\topskip is done if the columns start at the top of the page. \_createcolumns prepares columns with given height \_dime\_side by side to the \box1. \_printcolumns prints the columns prepared in \box1. The first \_box{} moves typesetting point to the next baseline. Next negative skip ensures that the first line from splitted columns is at this position.
\_setcolsize #1(\_dime=#1) \_relax

156
2.32 Citations, bibliography

2.32.1 Macros for citations and bibliography preloaded in the format

Registers used by \cite, \bib macros are declared here. The \bibnum counts the bibliography items from one. The \bibmark is used when \nonumcitations is set.

\bib 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\}.

\cite \{label\},...\{label\} manages \{labels\} using \_citeA 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 \{label\}\{text\} behaves like \rcite \{label\} but prints \{text\} instead \{bib-mark\}. The \{text\} is hyperlinked like \{bib-marks\} when \cite or \rcite is used. The empty internal macro \savedcites will include the \{bib-marks\} list to be printed. This list is set by \_citeA inside a group and it is used by \_printsavedcites in the same group. Each \cite/\rcite/\ecite macro starts from empty list of \{bib-marks\} because new group is opened.

\bibpart may be numbers or a special text related to cited bib-entry. It depends on \nonumcitations and on used bib-style. The mapping from \{label\} to \{bib-mark\} is done when \bib or \usebib is processed. These macros store the information to \_Xbib{\bibpart}{\{label\}}{\{number\}}{\{nonumber\}} where \{number\} and \{nonumber\} are two variants of \{bib-mark\} (numbered or text-like). This information is read from .ref file and it is saved to macros \bib{\bibpart}{\{label\}} and \bim{\bibpart}{\{number\}}. First one includes \{number\} and second one includes \{nonumber\}. The \lastbim{\bibpart} macro includes last number of bib-entry used in the document with given \{bibpart\}. A designer can use it to set appropriate indentation when printing the list of all bib-entries.
\_citeA\{label\} processes one label from the list of labels given in the parameter of \cite, \nocite, \rcite or \ecite macros. It adds the \{label\} to a global list \_ctlst:{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 save 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}\}k and we test this case by the command \if \bib:{number}\}k. 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 \_citeA macro runs repeatedly over the whole list of \{labels\}.

Because we implement possibility of more independent bibliography lists distinguished by \{bibpart\}, the \_addcitelist\{label\} macro must add the \{label\} to given \_ctlst:{bibpart}/.

When \_addcitelist is processed before \usebib, then \_citeI[\{label\}] is added. \usebib will use this list for selecting right records from .bib file. Then \usebib sets \_ctlst:{bibpart}/ to \_write. If \_addcitelist is processed after \usebib, then \Xcite[\{bibpart\}]/[\{label\}] is saved to the .ref file. The \Xcite creates \_ctlstB:{bibpart}/ as a list of saved \_citeI[\{label\}]. Finally, \usebib concatenates both lists \_ctlst:{bibpart}/ and \_ctlstB:{bibpart}/ in the second \TeX run.
The \bib marks (in numeric or text form) are saved in \_savedcites macro separated by commas. The \printsavedcites prints them by normal order or sorted if \sortcitations is specified or condensed if \shorcitations is specified.

The \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 \shortcitations sets simply \_lastcitenum=1. The macros for \bib marks printing follows (sorry, without detail documentation). They are documented in opmac-d.pdf (but only in Czech).

\begin{verbatim}
\def\printsavedcites{\_sortcitesA}
\chardef\tmpb=0 \ea\citeB\_savedcites,\fi}
\def\sortcitations{
\edef\savedcites{300000,\ea\_sortcitesB\_savedcites,}
\edef\tmpa####1300000,{\edef\savedcites{####1}}\ea\tmpa\_savedcites}
\def\sortcitesA{}
\def\sortcitations{
\edef\sortcitesA{\_edef\_savedcites{300000,\ea\_sortcitesB\_savedcites,}
\edef\tmpa####1300000,{\edef\_savedcites{####1}}\ea\tmpa\_savedcites}}
\def\citelinkA##1{\_trycs{_bim:\_bibp##1}{\_noexpand\nonumcitations + empty bibmark. Maybe bad bib-style}}}
\def\citelinkA{}
\def\nonumcitations{
\_lastcitenum=0 \def\sortcitesA{}
\def\etalchar##1{$^{##1}$}
\def\citelinkA##1{\_trycs{_bim:\_bibp##1}{\_noexpand\nonumcitations + empty bibmark. Maybe bad bib-style}}}
\def\shortcitations{\_lastcitenum=1}
\_public \nonumcitations \sortcitations \shortcitations ;
\end{verbatim}

The \bib {{\bibmark}} prints one bib-entry without reading any database. The bib-entry follows after this command. This command counts the used \bib counter and saves \_Xbib{\bibpart}{\_bibmark}{\_number}{\_nonumber} into .ref file immediately us-
ing \_wbib{(label)}{(number)}{(nonumber)}. This is the core of creation of mapping from \(\text{labels}\) to \(\text{number}\) and \(\text{nonumber}\).

\_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.

cite-bib.opm

\def\_bib{[\label\number\nonumber]}{\bibmark\bibB{[\label\number\nonumber]}\wbib{[\label\number\nonumber]}{\thebibnum}{\thebibmark}}

\def\_printbib \_ignorespaces {\par \_bibskip \_bibgl\_advance\bibnum by1 \_noindent \_def\_tmpb{[\label\number\nonumber]}\dbib{[\label\number\nonumber]}\wbib{[\label\number\nonumber]}{\thebibnum}{\thebibmark}\_printbib \_ignorespaces}

\_def\_dbib#1{\dest[\_bibp\thebibnum]\_printlabel{#1}}

\_def\_wbib#1#2#3{\if\wref\relax\else \immediate\_wref\_Xbib{\thebibpart}{#1}{#2}{#3}\fi\unless \ifcsname bib:\_bibp#1\endcsname \_Xbib{\thebibpart}{#1}{#2}{#3}\fi}

\let\_bibgl=\relax

\_public \bib ;

The \_printbib prints the bib-entry itself. You can re-define it if you want a different design. The \_pritbib 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 \(\langle\text{bib-marks}\rangle\) by \_llap{[\thebibnum]} If \_nonumcitations then the \_citelinkA is not empty and \(\langle\text{bib-marks}\rangle\) 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.

cite-bib.opm

\def\_printbib {\_hangindent=\_iindent\_if\_citelinkA\_empty \_hskip\_iindent \_llap{[\thebibnum]} \_fi}

\_def\_bibskip {\_ifnum\bibnum>0 \_smallskip \_fi}

\_def\usebib{\_par \_opinput {usebib.opm} \_usebib}

\_def\_usebib{
\_usebib}

\_public \_nobibwarning

\_nobibwarning [(\text{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.

\_def\_nobibwarnlist{,}

\_def\_nobibwarning[#1]{\global\_addto\_nobibwarnlist{#1,}}

\_public \_nobibwarning ;

\_def\nobibwarninglist{,}

\_def\_nobibwarning[#1]{\_global\_addto\_nobibwarnlist{#1,}}

\_public \_nobibwarning ;

2.32.2 The \_usebib command

The file usebib.opm implements the command \_usebib\langle\text{sorttype}\rangle\langle\text{style}\rangle\langle\text{bibfiles}\rangle where \(\langle\text{sorttype}\rangle\) is one letter \(c\) (references ordered by citation order in the text) or \(s\) (references ordered by key in the style file), \(\langle\text{style}\rangle\) is the part of the name \bib\langle\text{style}\rangle.opm of the style file and \(\langle\text{bibfiles}\rangle\) are one or more .bib file names without suffix separated by comma without space. Example:

\_usebib/s (simple) mybase,yourbase

This command reads the \(\langle\text{bibfiles}\rangle\) directly and creates the list of bibliographic references (only those declared by \_cite\[] or \_nocite\[] in the text). The formatting of such references is defined in the style file.

160
The principle “first entry wins” is used. Suppose \texttt{\usebib/s} \texttt{(simple)} \texttt{local, global}. If an entry with the same label is declared in \texttt{local.bib} and in \texttt{global.bib} too then the first wins. So, you can set exceptions in your \texttt{local.bib} file for your document.

The \texttt{bib-(style).opm} declares entry types (like \texttt{BOOK, ARTICLE}) and declares their mandatory and optional fields (like \texttt{author, title}). When a mandatory field is missing in an entry in the \texttt{.bib} file then a warning is printed on the terminal about it. You can suppress such warnings by command \texttt{\nobibwarning \{\bib-labels\}}, where \texttt{\bib-labels} is a comma-separated list of labels (without spaces) where missing mandatory fields will be no warned.

Old \texttt{.bib} files may use the obscure notation for accents like \texttt{\"o}. Recommendation: convert such old files to Unicode encoding. If you are unable to do this then you can set \texttt{\bibtexhook={\oldaccents}}.

### 2.32.3 Notes for bib-style writers

The \texttt{.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 \texttt{demo/op-biblist.bib} for a real example. The \texttt{\{entry-types\}} and \texttt{\{field-names\}} are case insensitive.

Ancient Bib\TeX has read such files and has generated files appropriate for reading by \TeX. It has worked with a set of \texttt{\{entry-types\}}, see the www page \url{http://en.wikipedia.org/wiki/BibTeX}. The set of entry types listed on this www page is de facto the Bib\TeX standard. The Op\TeX bib style writer must “declare” all such entry types and more non-standard entry types can be declared too if there is a good reason for doing it. The word “declare” used in the previous sentence means that a bib-style writer must define the printing rules for each \texttt{\{entry-type\}}. The printing rules for \texttt{\{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 \texttt{.bib} records.

The style writer can be inspired by two styles already done: \texttt{bib-simple.opm} and \texttt{bib-iso690.opm}. The second one is documented in detail in section 2.32.5.

The printing rules for each \texttt{\{entry-type\}} must be declared by \texttt{\_sdef\{\_print;\{entry-type\}\} in \texttt{bib-(style).opm} file. The \texttt{\{entry-type\}} has to be lowercase here. Op\TeX supports following macros for a more comfortable setting of printing rules:

- \texttt{\_bprinta} \texttt{\{field-name\}} \texttt{\{if defined\}} \texttt{\{if not defined\}}. The part \texttt{\{if defined\}} is executed if \texttt{\{field-name\}} is declared in \texttt{.bib} file for the entry which is currently processed. Else the part \texttt{\{if not defined\}} is processed. The part \texttt{\{if defined\}} can include the \texttt{*} parameter which is replaced by the value of the \texttt{\{field-name\}}.
- \texttt{\_bprintb} \texttt{\{field-name\}} \texttt{\{if defined\}} \texttt{\{if not defined\}}. The same as \texttt{\_bprinta}, but the \texttt{##1} parameter is used instead \texttt{*}. Differences: \texttt{##1} parameter can be used more than once and can be enclosed in nested braces. The \texttt{*} parameter can be used at most once and cannot be enclosed in braces. Warning: if the \texttt{\_bprintb} commands are nested (\texttt{\_bprintb} in \texttt{\_bprintb}), then you need to write the \texttt{#####1} parameter for internal \texttt{\_bprintb}. But if \texttt{\_bprinta} commands are nested then the parameter is not duplicated.
- \texttt{\_bprintc} \texttt{\_macro} \texttt{\{if non-empty\}}. The \texttt{\{if non-empty\}} part is executed if \texttt{\_macro} is non-empty. The \texttt{*} parameter can be used, it is replaced by the \texttt{\_macro}.
- \texttt{\_bprintv} \texttt{\{field1\},\{field2\}, \ldots} \texttt{\{if defined\}} \texttt{\{if not defined\}}. The part \texttt{\{if defined\}} is executed if \texttt{\{field1\}} or \texttt{\{field2\}} or ... is defined, else the second part \texttt{\{if not defined\}} is executed. There is one filed name or the list field names separated by commas. The parts cannot include any parameters.

There are two special field-names: \texttt{\_author} and \texttt{\_editor}. The processed list of authors or editors are printed here instead of raw data, see the commands \texttt{\_authorname} and \texttt{\_editorname} below.

The bib-style writer can define \texttt{\_print:BEGIN} and/or \texttt{\_print:END}. They are executed at the beginning or end of each \texttt{\{entry-type\}}. The formatting does not solve the numbering and paragraph indentation of the entry. This is processed by \texttt{\_printbib} macro used in Op\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
- \_namecont: the total number of the authors in the list
- \_Lastname, \_Firstname, \_Von, \_Junior: the parts of the name.

The whole style file is read in the group during the \usebib command is executed before typesetting the reference list. Each definition or setting is local here.

The auto-generated phrases (dependent on current language) can be used in bib-style files by \_mtext{bib.⟨identifier⟩}, where ⟨identifier⟩ is an identifier of the phrase and the phrase itself is defined by \_sdef{mt:bib.⟨identifier⟩}:⟨language⟩{(phrase)}. See section 2.37.2 for more detail. Phrases for ⟨identifiers⟩: and, etal, edition, citatedate, volume, number, prepages, postpages, editor, editors, available, availablealso, bachthesis, masthesis, phdthesis are defined already, see the end of section 2.37.2.

If you are using non-standard field-names in .bib database and bib-style, you have to declare them by \_CreateField {⟨fieldname⟩}.

You can declare \_SortingOrder in the manner documented by librarian package.

User or author of the bib-style can create the hidden field which has a precedence while sorting names. Example:

\_CreateField {sortedby}
\_SpecialSort {sortedby}

Suppose that the .bib file includes:

...  
  author = "Jan Chadima",
  sortedby = "Hzzadima Jan",
...

Now, this author is sorted between H and I, because the Ch digraph in this name has to be sorted by this rule.

If you need (for example) to place the auto-citations before other citations, then you can mark your entries in .bib file by sortedby = "@", because this character is sorted before A.

2.32.4 The \usebib.opm macro file loaded when \usebib is used

Loading the librarian.tex macro package. See texdoc librarian for more information about it.

We want to ignore \errmessage and we want not to create \jobname.lbr file.
Corrections in librarian macros.

```latex
\_emergencystretch=.3\_hsize
\_def\optexbibstyle{#2}\
\_setctable\_optexcatcodes
\_input bib-#2.omp
\_the \_bibtexhook
\_ifname \_mt:bib. and: \_cs{\_lan: \_the \_language} \_endsname \_else
\_opwarning{\_string\_usebib: No phrases for language}
\_cs{\_lan: \_the \_language} “(using “en”){}%
\_language=0 \_chardef\_documentlanguage=0
\fi
\def\tmp##1[*]##2\_relax{\def\tmp{##2}}\_ea\tmp\citelist[*]\_relax
\_ifx\tmp\empty\else% there was \nocite[*] used.
\_setbox0=\_vbox{\_hsize=\_maxdimen \_def\citelist{}\_adef@{\_readbibentry}\
\_input #3.bib\_ea}\_ea\def\ea\citelist\ea{\citelist}%
\fi
\def\citeI[##1]{\_csname lb@cite\_endcsname{##1}{\_bibp}{}{}}\citelist
\BibFile{#3}%
\_if s#1\_SortList{\_bibp}\_fi
\_ReadList{\_bibp}%
\restorectable
\_egroup
\fi
\} \_def\readbibentry#1#{\_readbibentryA}
\_def\readbibentryA#1{\_readbibentryB#1,,\relax!.}
\_def\readbibentryB#1#2,#3\_relax!.{\_addto\citelist{\citeI[#1#2]}}
```

usebib.omp
\def\lb@sorttype{#1#2}\
\fi
\lb@preparesortA#1#2\lb@eoe
\}
\def\_SpecialSort#1{\def\_sortfield{#1}}
\def\WriteImmediateInfo#1{} % the existence of .lbr file blocks new reading of .bib
\catcode`\@=\tmpnum

Main action per each entry.
\def\MakeReference{\par \_bibskip
\bbib\_advance\_bibnum by1
\_isdefined{\_bibp\_the\_bibnum}\_iftrue
\_edef\_tmpb{\_csname _bibp\_the\_bibnum\_endcsname}\%
\_bibmark=\ea{\_tmpb}\%
\_else \_bibmark={}\_fi
\_edef\_tmpb{\EntryKey}\
\noindent \_dbib\EntryKey
\_printbib
{\
\_RetrieveFieldIn{entrytype}\_entrytype
\_csname _print:BEGIN\_endcsname
\_isdefined{_print:\_entrytype}\_iftrue
\_csname _print:\_entrytype\_endcsname
\_else
\_ifx\_entrytype\_empty \_else
\_opwarning{Entrytype @\_entrytype\space from \[\EntryKey\] undefined}\%
\_csname _print:misc\_endcsname
\_fi\_fi
\_csname _print:END\_endcsname
\_wbib \EntryKey {\_the\_bibnum}{\_the\_bibmark}\%
\_par
}

The \_bprinta, \_bprintb, \_bprintc, \_bprintv commands used in the style files:
\def\_bprinta {\_bprintb*}
\def\_bprintb #1[#2#3]{\_def\_bibfieldname{#2#3}\
\_if!#2\relax
\_def\_bibfieldname{#3}\
\_RetrieveFieldIn{#3}\_bibfield
\_ifx\_bibfield\_empty\_else
\_RetrieveFieldIn(#3number)\_namecount
\_def\_bibfield{\_csname _Read#3\_ea\_endcsname \_csname _pp:#3\_endcsname}\%
\_fi
\_else
\_\_bibfield\_empty \_else
\_opwarning{Entrytype \_move\_entrytype\space from \[\EntryKey\] undefined}\%
\_csname _print:misc\_endcsname

\let\_Readauthor=\ReadAuthor \let\_Readeditor=\ReadEditor
\def\_bprintaA #1#2#3{#2#1#3}
\def\_bprintc#1#2{\_bprintcA#1#2**\relax}
\def\_bprintcA#1#2*#3*#4\relax{\_ifx#1\_empty \_else \_bprintcA#1#2**\_eeec\_fi}
\def\_bprintcB #1#2#3#4{\_ifx\_bibfield\_empty \_else \_bprintcA#1#2**\_eeec\_fi}
\let\_bprintcA\_bprintcA#1#2**\relax
\def\_bprintc\_bprintcA#1#2**\relax
\_ifx\_empty \_else \_bprintcA#1#2**\_eeec\_fi}
\def\_bprintv \[#1\]#2#3{\_def\_tmpa{#2}\_def\_tmpb{#3}\_bprintvA #1,,}
\def\_bprintvA #1,,{\_if^#1^#1\_else \_tmpb\_else \_tmpa\_def\_tmpa{}\_def\_tmpb{}\_fi
\_fi
\_if^#1^#1\_else \_tmpa\_else \_tmpb\_fi
\_fi
\_fi
\_if^#1^#1\_else \_tmpa\_else \_tmpb\_fi
\_fi
\_fi
\_fi
\_edef\_tmpb{\_csname _bibp\_the\_bibnum\_endcsname}\%
\_bibmark=\ea{\_tmpb}\%
\_else \_bibmark={}\_fi
\_edef\_tmpb{\EntryKey}\
\noindent \_dbib\EntryKey
\_printbib
{\
\_RetrieveFieldIn{entrytype}\_entrytype
\_csname _print:BEGIN\_endcsname
\_isdefined{_print:\_entrytype}\_iftrue
\_csname _print:\_entrytype\_endcsname
\_else
\_ifx\_entrytype\_empty \_else
\_opwarning{Entrytype @\_entrytype\space from \[\EntryKey\] undefined}\%
\_csname _print:misc\_endcsname
\_fi\_fi
\_csname _print:END\_endcsname
\_wbib \EntryKey {\_the\_bibnum}{\_the\_bibmark}\%
\_par
}
Various macros + multilingual. Note that \_nobibwarnlist is used in \_bibwarning and it is set by \nobibwarning macro.

2.32.5 Usage of the bib-iso690 style

This is the iso690 bibliographic style used by OpTEX.

See op-biblist.bib for an example of the .bib input. You can try it by:
\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} = \texttt{"value"}, or \texttt{name} = \texttt{\{value\}}. No matter which form is used. If the value is pure numeric then you can say simply \texttt{name} = \texttt{value}. Warning: the comma after each field value is mandatory! If it is missing then the next field is ignored or badly interpreted.

The entry names and field names are case insensitive. If there exists a data field no mentioned here then it is simply ignored. You can use it to store more information (abstract, for example).

There are “standard fields” used in ancient bibTEX (author, title, editor, edition, etc., see \texttt{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=\ldots}.

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):

- \texttt{Firstname(s) von Lastname}
- \texttt{von Lastname, Firstname(s)}
- \texttt{von Lastname, After, Firstname(s)}

Only the Lastname part is mandatory. Examples:

- Petr Olšák
- Olšák, Petr
- Leonardo Piero da Vinci
- da Vinci, Leonardo Piero
- da Vinci, painter, Leonardo Piero
The separator “ and ” between authors will be converted to comma during printing, but between the semifinal and final author the word “and” (or something different depending on the current language) is printed.

The first author is printed in reverse order: “LASTNAME, Firstname(s) von, After” and the other authors are printed in normal order: “Firstname(s) von LASTNAME, After”. This feature follows the ISO 690 norm. The Lastname is capitalized using uppercase letters. But if the \caps font modifier is defined, then it is used and printed \caps\_rm Lastname.

You can specify the option aumax:\langle number\rangle. The \langle number\rangle denotes the maximum authors to be printed. The rest of the authors are ignored and the et \textit{al}. is appended to the list of printed authors. This text is printed only if the auax value is less than the real number of authors. If you have the same number of authors in the .bib file as you need to print but you want to append et \textit{al}, then you can use auetal option.

There is an aumin:\langle number\rangle option which denotes the definitive number of printed authors if the author list is not fully printed due to auax. If aumin is unused then auax authors are printed in this case.

All authors are printed if auax:\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 \biboptions tokens list. For example:

\biboptions={auax:7 aumin:1}
% if there are 8 or more authors then only the first author is printed.

Examples:

\author = "John Green and Bob Brown and Alice Black",
output: GREEN, John, Bob BROWN, and Alice BLACK.

\author = "John Green and Bob Brown and Alice Black",
\option = "auax:1",
output: GREEN, John et al.

\author = "John Green and Bob Brown and Alice Black",
\option = "auax:2",
output: GREEN, John, Bob BROWN et al.

\author = "John Green and Bob Brown and Alice Black",
\option = "auax:3",
output: GREEN, John, Bob BROWN, and Alice BLACK.

\author = "John Green and Bob Brown and Alice Black",
\option = "auetal",
output: GREEN, John, Bob BROWN, Alice BLACK et al.

If you need to add a text before or after the author’s list, you can use the auprint:\{\langle value\rangle\} option. The \langle value\rangle will be printed instead of the authors list. The \langle value\rangle can include \AU macro which expands to the authors list. Example:

\author = "Robert Calbraith",
\option = "auprint:\{\AU\space [pseudonym of J. K. Rowling]\}"
output: CALBRAITH Robert [pseudonym of J. K. Rowling].

You can use the autrim:\langle number\rangle 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 \langle number\rangle. 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:
The editor field

The editor field is used for the list of the authors of the collection. The analogous rules as in author field are used here. It means that the authors are separated by “and”, the Firstnames, Lastnames, etc. are interpreted and you can use the options \texttt{edmax:<number>}, \texttt{edmin:<number>}, \texttt{edetal}, \texttt{edtrim:<number>} and \texttt{edprint:{<value>}} (with \texttt{\ED} macro). Example:

\begin{verbatim}
editor = "Jan Tomek and Petr Karas",
option = "edprint:{\ED, editors.} edtrim:1",
\end{verbatim}

Output: J. TOMEK and P. KARAS, editors.

If \texttt{edprint} option is not set then \texttt{\{\ED, eds.\}} or \texttt{\{\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 editional info. The value is read as raw data without any interpretation of Lastname, Firstname etc.

\begin{verbatim}
ednote = "Illustrations by Robert Agarwal, edited by Tom Nowak",
\end{verbatim}

Output: Illustrations by Robert AGARWAL, edited by Tom NOWAK.

The \texttt{upper} command has to be used for Lastnames in the 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 \texttt{titlepost:{<value>}}. Example:

\begin{verbatim}
title = "The Simple Title of The Work",
or
    title = "Main Title: Subtitle",
or
    title = "Main Title: Subtitle",
    option = "titlepost:{Secondary title}"
\end{verbatim}

The output of the last example: \textit{Main Title: Subtitle}. Secondary title.

The edition field

This field is used only for second or more edition of cited work. Write only the number without the word "edition". The shortcut "ed." (or something else depending on the current language) is added automatically. Examples:

\begin{verbatim}
edition = "Second",
edition = "2nd",
edition = "2\$\{\rm nd\}$",
edition = "2."
\end{verbatim}

Output of the last example: 2. ed.

\begin{verbatim}
edition = "2."
lang = "cs",
\end{verbatim}

Output: 2. vyd.

Note, that the example \texttt{edition=Second} may cause problems. If you are using language "cs" then the output is bad: Second vyd. But you can use \texttt{editionprint:{<value>}} option. The the \texttt{<value>} is printed instead of edition field and shortcut. The edition field must be set. Example:

\begin{verbatim}
edition = "whatever",
optioon = "editionprint:{Second full revised edition}"
\end{verbatim}

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
definition = "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:{\langle value\rangle}" option. If it is set then the \langle value\rangle 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 \url 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: [vidl. 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
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 @MISC entry

It is intended for various usage.

Fields: author, title, howpublished, ednote, citedate, doi, url, note.

You can use \AU, \ED, \EDN, \VOL, \NO, \PP, \ADDR, \PUBL, \YEAR macros in ednote field. These macros print authors list, editors list, edition, volume, number, pages, address, publisher, and year field values respectively.

The reason for this entry is to give to you the possibility to set the format of entry by your own decision. The most of data are concentrated in the ednote field.

• The @BOOKLET, @INCOLLECTION, @MANUAL, @PROCEEDINGS, @TECHREPORT, @UNPUBLISHED entries

These entries are equivalent to @MISC entry because we need to save the simplicity. They are implemented only for (almost) backward compatibility with the ancient BibTeX. But the ednote is mandatory field here, so you cannot use these entries from the old databases without warnings and without some additional work with the .bib file.

The cite-marks (bibmark) used when \nnumcitations is set

When \nnumcitations 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 \nnumcitations is set.

If you have some problems with name sorting, you can use the hidden field key, which is used for sorting instead of the “Lastname Firstname(s)” of authors. If the key field is unset then the “Lastname Firstname(s)” is used for sorting normally. Example:

author = "Světla Čmejrková",
key = "Czzmejrkova Svetla",

This entry is now sorted between C and D.

The norm recommends placing the auto-citations at the top of the list of references. You can do this by setting key=\"@\", 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 \text{bib.(identifier)} if you want to use a phrase dependent on outer document language (no on entry language). Example:

```latex
\texttt{howpublished = "\text{bib.blue-ray}"}
```

Now, you can set the variants of bib.blue-ray phrase for various languages:

```latex
\_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 bib\TeX: 

- 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.

### 2.32.6 Implementation of the bib-iso690 style

\begin{verbatim}
\_maybetod (alias \: in the style file group) does not put the second dot.
\end{verbatim}

\begin{verbatim}
\_maybedot {\_ifnum\spacefactor=\sfcode\,.\relax\else.\fi}
\end{verbatim}

Option field.
Formating of Author/Editor lists.

Preparing bib-mark (used when \nonumcitations is set).
Setting phrases.

Non-standard field names.

Sorting.

Supporting macros.
Entry types.

Entry types.

Entry types.

Entry types.
2.33 Sorting and making Index

```latex
\begin{verbatim}
\_sdef{_print:thesis}{%  
\_bprintb [author] {\_doauthor1{##1}\: \{\_bibwarning}\%  
\_bprintb [title] {{\_em##1}}\_bprintc\_titlepost{\: \*} \_bprintv[howpublished]\{\:\}\%  
\_bookgeneric{\_bprintb [pages] \{\_preprint\_hbox{##1}.\ \}}%  
\_slet{_print:phdthesis}{\_def\_thesistype{\_Mtext{bib.phdthesis}}} \_cs{_print:thesis}}%  
\_slet{_print:mastershesis}{\_def\_thesistype{\_Mtext{bib.masthesis}}} \_cs{_print:thesis}}%  
\_slet{_print:bachelorsthesis}{\_def\_thesistype{\_Mtext{bib.bachthesis}}} \_cs{_print:thesis}}%  
\_slet{_print:booklet}{\_print:generic}  
\_slet{_print:incollection}{\_print:generic}  
\_slet{_print:manual}{\_print:generic}  
\_slet{_print:proceedings}{\_print:generic}  
\_slet{_print:unpublished}{\_print:generic}  
\_slet{_print:misc}{\_let\_bibwarning=\_relax \_cs{_print:generic}}%  
\end{verbatim}
```

3 \_codedecl \makeindex {Makeindex and sorting <2022-06-28>} \% preloaded in format

175
\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 diéresis 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 \texttt{\_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 \texttt{\_sortingdata...} macro. Others not mentioned characters have undefined behavior during sorting.

Sorting is always processed by rules of a given language. The macros \texttt{\_sortingdata(lang-tag)}, \texttt{\_ignoredchars(lang-tag)} and \texttt{\_compoundchars(lang-tag)} declare these rules. The \langle lang-tag \rangle 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 \texttt{lang-data.opm} file (see section 2.37.4).
The \_compoundchars\{lang-tag\} can declare changes performed before sorting. For example Czech language declares:

\_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 \textit{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:

\_let \_sortingdatade = \_sortingdatalatin % German alphabet is subset of Latin
\_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:

\_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:


Because u < ü in \_sortingdata and because \_xcompoundcharsde is empty, we have Mueller < 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 ^^^I, ^^^J or ^^^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.opm 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.opm 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 \_\{string\} control sequences (to save \TeX\ memory). The \_\{preparesorting \_\{string\}\} converts \_\{string\} to \_\texttt{tmpb} with respect to the data initialized in \_\{setprimarysorting\} or \_\{setsecondarysorting\}. The compound characters are converted by the \_\{docompound\} macro.

Macro \_\{isAleB \_\{string1\}\,\_\{string2\}\} returns the result of comparison of given two strings to \_\{ifAleB\} control sequence. Usage: \_\{isAleB \_\{string1\}\,\_\{string2\}\,\_\{ifAleB\}, \_\{ifAleB\}, \_\{ifAleB\} ... \_\{else\} ... \_\{fi\}. The converted strings (in respect of the data prepared for first pass) must be saved as values of \_\{string1\} and \_\{string2\} macros. The reason is speed: we don’t want to convert them repeatedly in each comparison. The macro \_\{testAleB\} \_\{converted-string1\}\&\_\{relax\}\{converted-string2\}\&\_\{relax\} \_\{string1\}\,\_\{string2\}\ does the real work. It reads the first character from both converted strings, compares them and if it is equal then calls itself recursively else gives the result.

The \_\{testAleBsecondary\} \_\{string1\}\,\_\{string2\}\ 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 \_\texttt{tmpa} and \_\texttt{tmpb} and corrects them by \_\{prepsecondarypass\} if needed. Finally, the test is recursively done by the macro \_\{testAleBsecondaryX\} \_\{converted-string1\}\_\{relax\}\{converted-string2\}\_\{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 \dosing macro must prepare commas between items.

The \dosing \list macro redefines \list as sorted \list. The \list have to include control sequences in the form \langle c \rangle \langle string \rangle. These control sequences will be sorted with respect to \langle strings \rangle without change of meanings of these control sequences. Their meanings are irrelevant when sorting. The first character \langle c \rangle in \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 \ldots.

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 \asciiortingtrue then ASCII sorting will be processed and all language sorting data will be ignored.
French rules need 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 \langle word \rangle prints one item to the index. If \_\iis \langle word \rangle is defined then this is used instead real \langle word \rangle (this exception is declared by \iis macro). Else \langle word \rangle is printed by \_printii.

Finally, \_printiipages prints the value of \langle word \rangle, i.e. the list of pages.
You can re-define \_pgprint \langle pageno \rangle: \{\langle iitype \rangle \} if you need to implement more \{iitypes\}.

The \_\index{\langle word \rangle} puts one \langle word \rangle to the index. It writes \_\Xindex{\langle word \rangle}{\langle iitype \rangle} to the .ref file. All other variants of indexing macros expand internally to \_\index.

The \_\index{\langle word \rangle}{\langle iitype \rangle} stores \langle word \rangle to the \_iilist if there is the first occurrence of the \langle word \rangle. The list of pages where \langle word \rangle occurs, is the value of the macro \_\langle word \rangle, so the
The implementation of macros \ii, \iid, \iis follows. Note that \ii works in the horizontal mode in order to the \write whatis is not broken from the following word. If you need to keep vertical mode, use \iiindex{(word)} directly.

The \iitype{(type)} saves the \(type\) to the \_iitypesaved macro. It is used in the \iiindex macro.

\section{Footnotes and marginal notes}

\gfnotenum is a counter which counts footnotes globally in the whole document.
\lfnotenum is a counter which counts footnotes at each chapter from one. It is used for local page footnote counters too.
\ifpgfnote says that footnote numbers are counted on each page from one. We need to run \openref in this case.
\fnotenum is a macro that expands to footnote number counted in declared part.
\fnotenumchapters declares footnotes numbered in each chapter from one (default), \fnotenumglobal declares footnotes numbered in whole document from one and \fnotenumpages declares footnotes numbered at each page from one.
The \printfnotemark prints the footnote mark. You can re-define this macro if you want another design of footnotes. For example:

```latex
\fnotenumpages
\def \printfnotemark {
\ifcase 0\fnotenum\or *
\or **\or ***\or $^{\mathbox{†}}$\or $^{\mathbox{‡}}$\or $^{\mathbox{††}}$i
\}
\def \printfnotemarkA {
\link[fnt:\the\gfnotenum]{\printfnotemarkA}
\dest[fnf:\the\gfnotenum]
\}
\def \printfnotemarkB {
\link[fnf:\the\gfnotenum]{\printfnotemarkB}
\dest[fnt:\the\gfnotenum]
\}
\public \fnotelinks ;
```

Each footnote saves the \Xfnote (without parameter) to the .ref file (if \openref). We can create the mapping from \gfnotenum to \pgfnotenum in the macro \fn. Each \Xpage macro sets the \lfnotenum to zero.

The \fnote \langle text \rangle macro is simple, \fnotemark and \fnotetext does the real work.

The \fnotetext calls \opfootnote which is equivalent to plain \TeX \vfootnote. It creates new data to Insert \footins. The only difference is that we can propagate a macro parameter before the text is printed (see section 2.18). This propagated macro is \fnset which sets smaller fonts.

Note that \vfootnote and \opfootnote don’t read the text as a parameter but during the normal horizontal mode. This is the reason why catcode changes (for example in-line verbatim) can be used here.

By default \mnote \langle text \rangle 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: \langle mnotenum \rangle as \right or \left when the \.ref file is read. The \ifnum 0\mnotenum<#2 \trick returns true if \pageno has a numeric type and false if it is a non-numeric type (Roman numeral, for example). We prefer to use \pgpageno, but only if it has the numeric type. We use \gppageno in other cases.
User can declare \fixmnotes\left or \fixmnotes\right. It defines \_mnotesfixed as \_left or \_right which declares the placement of all marginal notes and such declaration has a precedence.

The \_mnoteD\{⟨text⟩\} macro sets the position of the marginal note. The outer box of marginal note has zero width and zero depth and it is appended after current line using \vadjust primitive or it is inverted to vertical mode as a box shifted down by \parskip and with \vskip\=\baselineskip followed.

The \_mnoteskip is a dimen value that denotes the vertical shift of marginal note from its normal position. A positive value means shift up, negative down. The \_mnoteskip register is set to zero after the marginal note is printed. The new syntax \mnote up\{dimen\}\{⟨text⟩\} is possible too, but public \_mnoteskip is kept for backward compatibility.

We don’t want to process \fnote, \fnotemark, \mnote in TOC, headlines nor outlines.

2.35 Styles
OpteX provides three styles: \report, \letter and \slides. Their behavior is documented in user part of the manual in the section 1.7.2 and \slides style (for presentations) is documented in op-slides.pdf which is an example of the presentation.

2.35.1 \report and \letter styles
The \report style initialization macro is defined here.

The \letter style initialization macro is defined here.

The \slides macro reads macro file \slides.opm, see the section 2.35.2.

2.35.2 \slides style for presentations

Default margins and design is declared here. The \ttfont is scaled by mag1.15 in order to balance the ex height of Helvetica (Heros) and LM fonts Typewriter. The \begtt...\endtt verbatim is printed by smaller text.
The bottom margin is set to 3 mm. If we use 1 mm, then the baseline of \footline is 2 mm from the bottom page. This is the depth of the Grey rectangle used for page numbers. It is r-lapped to \hoffset width because left margin = \hoffset = right margin. It is 14 mm for narrow pages or 16 mm for wide pages.

The \subtit is defined analogically like \tit.

The \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 \slideshow is active then each page is opened by \setbox\_slidepage=\vbox\bgroup (roughly speaking) and closed by \egroup. The material is \unvboxed and saved for the usage in the next usage if \pg+ is in process. The \_slidelayer is incremented instead \pageno if \pg+. This counter is equal to \count1, so it is printed to the terminal and log file next to \pageno.

The code is somewhat more complicated when \layers is used. Then \langle layered-text \rangle is saved to the \_layertext macro, the material before it is in \_slidepage box and the material after it is in \_slidepageB box. The pages are completed in the \loop which increments the \_layernum register and prints page by the \_printlayers
When \texttt{\textbackslash 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 \texttt{\textbackslash 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 \texttt{\textbackslash slidetext} macro. We can move creating these destinations to the output routine. \texttt{\textbackslash destbox} is saved value of the original \texttt{\textbackslash destbox} which is redefined to do only \texttt{\addto\destboxes\textbackslash destbox[\texttt{\string\textbackslash label}]}. All destinations saved to \texttt{\textbackslash destboxes} are created at the start of the next output routine in the \texttt{\pagedest} macro. The output routine removes \texttt{\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 \texttt{\textbackslash label} was used. The first “virtual” page where \texttt{\textbackslash label} is used includes its destination. If you want to go to the final page of the partially uncovering ideas then use \texttt{\textbackslash label[\texttt{\string\textbackslash label}]\texttt{\textbackslash text}} in the last part of the page (before \texttt{\pg;}) or use \texttt{\pgref} instead \texttt{\ref}.

The \texttt{\textbackslash settinglayer} is used in the \texttt{\layertext} macro to prevent printing “Duplicate label” warning when it is expanded. It is done by special value of \texttt{\slideshook} (used by the \texttt{\label} macro). Moreover, the warning about illegal use of \texttt{\bib}, \texttt{\usebib} in \texttt{\layers} environment is activated.

Default \texttt{\textbackslash layers(\texttt{num})} macro (when \texttt{\textbackslash slideshow} is not activated) is simple. It prints the \texttt{\langle\texttt{layered-text}\rangle} with \texttt{\layernum=\langle\texttt{num}\rangle+1} because we need the result after last layer is processed.
We must redefine \texttt{fnotenumpages} because the data from \texttt{.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}
\def \fnotenumpages{\def \fnotenum{\the \lfnotenum}\pgfnotefalse
\def \lfnotenumreset{\global \lfnotenum=0}}
\let \lfnotenumreset=\relax
\public \fnotenumpages ;
\end{verbatim}

\section{Logos}

Despite plain \TeX{} each macro for logos ends by \texttt{\ignoreslash}. This macro ignores the next slash if it is present. You can use \texttt{\TeX/ like this} for protecting the space following the logo. This is visually more comfortable. The macros \texttt{\TeX{}}, \texttt{\OpTeX{}}, \texttt{\LuaTeX{}}, \texttt{\XeTeX{}} are defined.

\begin{verbatim}
\protected\def \TeX{\TeX\ignoreslash}
\protected\def \OpTeX{Op\ignoreslash \TeX}
\protected\def \LuaTeX{Lua\ignoreslash \TeX}
\protected\def \XeTeX{X\ignoreslash E% \pdfsave\rlap{\pdfscale{-1}{1}\lower.5ex\hbox{E}}\pdfrestore \TeX}
\def \ignoreslash{\ifx/\else \fi}
\regmacro {}{}{% conversion for PDF outlines
\def \TeX{TeX\ignoreslash}\def \OpTeX{Op\ignoreslash \TeX}\def \LuaTeX{Lua\ignoreslash \TeX}\def \LaTeX{LaTeX\ignoreslash}\def \OPmac{OPmac\ignoreslash}\def \CS{CS}\def \csplain{csplain\ignoreslash}}
\public \ignoreslash ;
\end{verbatim}

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
\raise \tmpdim \hbox{\thefontscale[710]A}%
\kern-.15em \kern-\slantcorr \TeX}
\def \slantcorr{\ea \ignorept \the \fontdimen1 \font \tmpdim}
\public \LaTeX ;
\end{verbatim}

\OPmac, \CS{} and \csplain logos.

\begin{verbatim}
\def \OPmac{\leavevmode
\lower.2ex \hbox{\thefontscale[1400]0}\kern-.86em P{\em mac}\ignoreslash}
\def \CS{$\cal C$\ignoreslash}
\def \csplain{csplain\ignoreslash}
\public \OPmac \CS \csplain ;
\end{verbatim}

The expandable versions of logos used in Outlines need the expandable \texttt{\ignnslash} (instead of the \texttt{\ignoreslash}).

\begin{verbatim}
\regmacro {}{}{\% conversion for PDF outlines
\def \TeX{TeX\ignoreslash}\def \UpTeX{Up\ignoreslash \TeX}\def \LuaTeX{Lua\ignoreslash \TeX}\def \XeTeX{Xe\ignoreslash \TeX}\def \OPmac{OPmac\ignoreslash}\def \CS{CS}\def \csplain{csplain\ignoreslash}}
\public \ignnslash ;
\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 \lccode and \uccode. The \catcode above the code 127 is not set, i.e. the \catcode=12 for all codes above 127.

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. 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 file uni-lcuc.opm is left by \endinput.

If the file UnicodeData.txt does not exist then internal data are used. They follow to the end of the file uni-lcuc.opm.

\begin{verbatim}
\_isfile{UnicodeData.txt}\_iftrue
\_begingroup
\_sdef{lc:\Ll}{#1#2#3#4}{\_global\_lccode"#2="#2 \_global\_uccode"#2="0#3 }
\_sdef{lc:\Lu}{#1#2#3#4}{\_global\_lccode"#2=0#4 \_global\_uccode"#2="#2 }
\_sdef{lc:\Lo}{#1#2#3#4}{\_global\_lccode"#2="#2 \_global\_uccode"#2="#2 }
\_def\_pa#1#2#3#4#5#6#7#8#9;{\_ifx;#1;\_else\_ea\_pb\_fi{#1}{#3}}
\_def\_pb#1#2#3#4#5#6#7#8;{\csname lc:#2\_endcsname\_pc{#1}{#6}{#7}\_pa}
\_def\_pc#1#2#3{} % ignored if the character hasn't Ll, Lu, nor Lo type
\_everyeof={;;;;;;;;;} % end of file
\_ea\_pa\_input UnicodeData.txt
\_endgroup \_endinput \_fi % \endinput here, if UnicodeData.txt was loaded
\end{verbatim}

If UnicodeData.txt not found, we have internal copy here from csplain, 2014:

\begin{verbatim}
\_isfile{UnicodeData.txt}\_iftrue
\_begingroup
\_sdef{lc:\Ll}{#1#2#3#4}{\_global\_lccode"#2="#2 \_global\_uccode"#2="0#3 }
\_sdef{lc:\Lu}{#1#2#3#4}{\_global\_lccode"#2=0#4 \_global\_uccode"#2="#2 }
\_sdef{lc:\Lo}{#1#2#3#4}{\_global\_lccode"#2="#2 \_global\_uccode"#2="#2 }
\_def\_pa#1#2#3#4#5#6#7#8#9;{\_ifx;#1;\_else\_ea\_pb\_fi{#1}{#3}}
\_def\_pb#1#2#3#4#5#6#7#8;{\csname lc:#2\_endcsname\_pc{#1}{#6}{#7}\_pa}
\_def\_pc#1#2#3{} % ignored if the character hasn't Ll, Lu, nor Lo type
\_everyeof={;;;;;;;;;} % end of file
\_ea\_pa\_input UnicodeData.txt
\_endgroup \_endinput \_fi % \endinput here, if UnicodeData.txt was loaded
\end{verbatim}

...etc., 15900 similar lines (see uni-lcuc.opm)

2.37.2 Multilingual phrases and quotation marks

Four words are generated by OpT\eX\ macros: “Chapter”, “Table”, “Figure” and “Subject”. These phrases are generated depending on the current value of the \language register, if you use \_mtext{⟨phrase-id⟩}, specially \_mtext{chap}, \_mtext{t}, \_mtext{f} or \_mtext{subj}. If your macros generate more words then you can define such words by \sdef{\_mt:⟨phrase-id⟩}{⟨lang-tag⟩} where ⟨phrase-id⟩ is a label for the declared word and ⟨lang-tag⟩ is a language shortcut declared by \_preplang.

\begin{verbatim}
\_def\_mtext#1{\_trycs{\_mt:#1:\_trycs{\lan:\_the\_language}{en}}
\_csname _mt:#1:en\_endcsname}}
\end{verbatim}

languages.opm
We can declare such language-dependent words by
\_sdef\_mt:chap:en\{Chapter\} \_sdef\_mt:chap:cs\{Kapitola\}
\_sdef\_mt:en\{Table\} \_sdef\_mt:cs\{Tabulka\}
e tc. but we use more “compact” macro \_lang\lang-tag\_chapter\_table\_figure\_subject for declaring them.

\_def \_langu #1 #2 #3 #4 #5 {%
  \_sdef\_mt:chap:en\{#2\}\_sdef\_mt:tt:en\{#3\}\_sdef\_mt:tt:cs\{#4\}%
  \_sdef\_mt:subj:en\{#5\}%
}

More phrases are auto-generated in bibliography references. They are declared by
\_langb\lang-tag\{(and)\} \{(et-al)\} \{(cit)\} \{(vol)\} \{(no)\} \{(pp)\} \{(ed)\} \{(eds)\}
\{(avail-from)\} \{(avail-to)\} \{(ba-thesis)\} \{(ma-thesis)\} \{(phd-thesis)\}. It is used similar way as the
\_langw above. Both these macros are used in lang-data.opm file, see the end of section 2.37.3.

\_today macro needs auto-generated words for each name of the month.
\_monthw\lang-tag\{January\} \{February\} \{March\} \{April\} \{May\} \{June\} \{July\} \{August\} \{September\} \{October\} \{November\} \{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 \{lang-tag\} is en.

Quotes should be tagged by \"\text\" and \’\text\’ if \{iso-code\}quotes is declared at beginning of the document (for example \quoutes). If not, then the control sequences \" and \’ are undefined. Remember, that they are used in another meaning when the \oldaccents command is used. The macros \" and \’ are not defined as \protected because we need their expansion when \outlines are used. User can declare quotes by \quotes and \csquotes \frquotes \dequotes \skquotes \quoteschars. or use \altquotes to swap between the meaning of these two types of quotes. \quoutes, \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\{clqq\} \{crqq\} \{clq\} \{crq\} in the lang-data.opm file.

\_def \_quotationmarks \{\quotemarks\lang-tag\{clqq\} \{crqq\} \{clq\} \{crq\}\}

\_public \today ;

191
The \quoteschars⟨lqq⟩⟨rq⟩⟨lq⟩⟨rq ⟩ defines \" and \" as \_qqA\_qqB in normal mode and as expandable 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 \"{"}\. This is the reason why the sub-verbatim mode is used when the first char is { in the parameter.

The \~\" is defined as \_qqA\_qqB⟨lqq⟩⟨rq⟩ and \~\" as \_qqA\_qqC⟨lq⟩⟨rq⟩. The \_qqA\_qqB⟨clqq⟩⟨crqq⟩ runs \_qqB⟨lqq⟩⟨rq⟩ (text)\". The \_regquotes \~\"⟨L⟩⟨R⟩ does \def \#1\{⟨L⟩\#1⟨R⟩\} for outlines but the " separator is active (because " and ' are active in \pdfunidef).
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 TeXlive are declared here. The language switches \cslang, \sklang, \delang, \pllang and many others are declared. You can declare more languages by \_preplang in your document, if you want.

The usage of \_preplang with ⟨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 ⟨lang-id⟩lang switch.
<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Galician</td>
<td>gl</td>
<td>gl</td>
</tr>
<tr>
<td>Georgian</td>
<td>ka</td>
<td>ka</td>
</tr>
<tr>
<td>Gujarati</td>
<td>gu</td>
<td>gu</td>
</tr>
<tr>
<td>Hindi</td>
<td>hi</td>
<td>hi</td>
</tr>
<tr>
<td>Indonesian</td>
<td>id</td>
<td>id</td>
</tr>
<tr>
<td>Interlingua</td>
<td>ia</td>
<td>ia</td>
</tr>
<tr>
<td>Kannada</td>
<td>kn</td>
<td>kn</td>
</tr>
<tr>
<td>Kurmanji</td>
<td>kmr</td>
<td>kmr</td>
</tr>
<tr>
<td>Malayalam</td>
<td>ml</td>
<td>ml</td>
</tr>
<tr>
<td>Marathi</td>
<td>mr</td>
<td>mr</td>
</tr>
<tr>
<td>Mongolian</td>
<td>mn</td>
<td>mn-cyril</td>
</tr>
<tr>
<td>Nynorsk</td>
<td>nn</td>
<td>nn</td>
</tr>
<tr>
<td>Occitan</td>
<td>oc</td>
<td>oc</td>
</tr>
<tr>
<td>Oriya</td>
<td>or</td>
<td>or</td>
</tr>
<tr>
<td>Pali</td>
<td>pi</td>
<td>pi</td>
</tr>
<tr>
<td>Panjabi</td>
<td>pa</td>
<td>pa</td>
</tr>
<tr>
<td>Piedmontese</td>
<td>pms</td>
<td>pms</td>
</tr>
<tr>
<td>Pinyin</td>
<td>zh</td>
<td>zh- fancypinyin</td>
</tr>
<tr>
<td>Sanskrit</td>
<td>sa</td>
<td>sa</td>
</tr>
<tr>
<td>Tamil</td>
<td>ta</td>
<td>ta</td>
</tr>
<tr>
<td>Telugu</td>
<td>te</td>
<td>te</td>
</tr>
<tr>
<td>Thai</td>
<td>th</td>
<td>th</td>
</tr>
<tr>
<td>Turkish</td>
<td>tr</td>
<td>tr</td>
</tr>
<tr>
<td>Turkmen</td>
<td>tk</td>
<td>tk</td>
</tr>
<tr>
<td>Upper Sorbian</td>
<td>hsb</td>
<td>hsb</td>
</tr>
</tbody>
</table>

The `\_preplangmore {lang-id}{space}{text}` declares more activities of the language switch. The `{text}` is processed whenever `\_langspecific:lang-id(lang)` is invoked. If `\_preplangmore` is not declared for a given language then `\_langdefault` is processed.

You can implement selecting a required script for given language, for example:

```
\_preplangmore ru {\_frenchspacing \_setff{script=cyril}\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 default `\_language=0` is US-English with original hyphenation patterns preloaded in the format (see the end of section 2.10). We define `\_enlang` and `\_enlang` switches. Note that if no language switch is used in the document then `\_language=0` and US-English patterns are used, but `\_frenchspacing` isn't set.

```
\chardef\_enPatt=0
\_sdef{\lan:0}{en}
\_sdef{\ulan:english}{en}
\_def\_enlang{\_uselang(en)\_enPatt23} % \lefthyph=2 \righthyph=3
\_def\_enlang{\_uselang(en)\_enPatt23}
```

The list of declared languages are reported during format generation.

```
\_message{Declared languages: \_langlist.}
```

Each language switch `\_langspecific:lang-id(lang)` defined by `\_preplang` has its initial state `\_langinit {switch} {lang-id}{{LongName}}{{lang-tag}{{{LAPHYP}}}{{b-laphyn}}}`. The `\_langinit` macro does:
• The internal language \langle number \rangle is extracted from \texttt{\_the\:\langle lang-id\rangle Patt}.

• \texttt{\_def \_lan:\langle number \rangle \{\langle lang-tag \rangle\}} for mapping from \texttt{\_language=\langle number \rangle}.

• loads \texttt{\_langinput} for \texttt{\_language=\langle lang-tag \rangle}.

• \texttt{\_def \langle lang-id\rangle \_uselang\{\langle lang-id\rangle\} \langle \_lan: \langle number \rangle \{\langle lang-tag \rangle\}\}} i.e. the switch redefines itself for doing a “normal job” when the language switch is used repeatedly.

• Runs itself (i.e. \texttt{\_\langle lang-id\rangle lang}) again for doing the “normal job” firstly.

\begin{verbatim}
\_def\_langinit \#1\#2\#3\#4\#5\#6\#7{% \\_switch lang-id(LongName)lang-tag[hyph-file]lr-hyph
\_sxdef{_lan:\_ea\_the\_csname _#2Patt\_endcsname}{#4}%
\_begingroup \_setbox0=\_vbox{% we don't want spaces in horizontal mode
\_setctable_optexcatcodes
\% loading patterns:
\_language=\cs_copy{\language}{\_#2Patt}\_relax
\_ifx^#5^\_else
\_wlog{Loading hyphenation for #3: \_string\language=\_the\_language\_space(#5)}%
\_let\patterns=\_patterns \_let\hyphenation=\_hyphenation \_def\message##1{}%
\_isfile {hyph-#5}\_iftrue \_input{hyph-#5}%
\_else \_opwarning{No hyph. patterns #5 for #3, missing package?}\_fi
\_fi
\% loading language data:
\_langinput{#4}%
\xdef#1{\noexpand\_uselang{#2}\_csname _#2Patt\_endcsname #6#7}%
\_do language switch
}
\_uselang{\langle lang-id\rangle} \langle \_lan: \langle number \rangle \{\langle lang-tag \rangle\}\}} is used as “normal job” of the switch. It sets \texttt{\_language}, \texttt{\_lefthyphenmin}, \texttt{\_righthyphenmin}. Finally, it runs data from \texttt{\_preplangmore} or runs \texttt{\_langdefault}.

\begin{verbatim}
\_def\_uselang#1#2#3#4{
\_language=#2\_lefthyphenmin=#3\_righthyphenmin=#4\_relax
\_langreset \_def\_langreset{}\_trycs{_langspecific:#1}{\_langdefault}%
}
\end{verbatim}

The \texttt{\_uselanguage \{\langle LongName \rangle\}} macro is defined here (for compatibility with e-plain users). Its parameter is case insensitive.

\begin{verbatim}
\_def\_uselanguage#1{\_def\_tmp{#1}\_lowercase{\_cs{\_trycs{\_ulan:#1}{0x}lang}}}
\_sdef{_0xlang}{\_opwarning{\_string\uselanguage{\_tmp}: Unknown language name, ignored}}
\_public \uselanguage ;
\end{verbatim}

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 are read by \texttt{\_langinput{\langle lang-tag \rangle}}. First few lines of the file looks like:

\begin{verbatim}
\_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.} {\_ed.} {\_cit.-} {\_Vol.-} {\_No.-} {\_pp.-} {\_p.} {\_-.ed.} {\_-.eds.}
\{Available from } {Available also from }
\{Bachelor's Thesis\} {\_Master's Thesis\} {\_Ph.D. Thesis\}
\_monthw en January February March April May June
\_monthw en July August September October November December
\_months{\_mt:today:en}{\_mtext{m\_the\_month} \_the\_day, \_the\_year}
\_quotationmarks en {""''}
\%\_let \_sortingdataen = \_sortingdatalatin % set already, see section 2.33, makeindex.opm
\_let \_ignoredcharsen = \_ignoredcharsgeneric
\_def \_compoundcharsen {}
\_langdata cs (Czech) %  
\end{verbatim}
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 \nolanginput{⟨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.

Data of two preferred languages are preloaded in the format:
2.38 Other macros

Miscellaneous macros are here.

\texttt{\textbackslash useOpTeX} and \texttt{\textbackslash useoptex} are declared as \texttt{\textbackslash relax}.

\begin{verbatim}
\let \useOpTeX = \relax \let \useoptex = \relax
\end{verbatim}

The \texttt{\textbackslash lastpage} and \texttt{\textbackslash totalpages} get the information from the \texttt{\_currpage}. The \texttt{\_Xpage} from .ref file sets the \texttt{\_currpage}.

\begin{verbatim}
\def \_totalpages {\_openref\_ea\_ignoresecond\_currpage}
\def \_lastpage {\_openref\_ea\_usesecond\_currpage}
\def \_currpage {{0}{?}}
\end{verbatim}

We need \texttt{uv}, \texttt{\textbackslash lq}, \texttt{\textbackslash rq}, \texttt{\textbackslash flq}, \texttt{\textbackslash frq}, \texttt{\textbackslash uslang}, \texttt{\textbackslash ehyph} \texttt{\textbackslash chyph}, \texttt{\textbackslash shyph}, for backward compatibility with \texttt{CSplain}. Codes are set according to Unicode because we are using Czech only in Unicode when \texttt{LuaTeX} is used.

\begin{verbatim}
\letfont was used in CSplain instead of \texttt{\fontlet}.
\end{verbatim}

Non-breaking space in Unicode.

\begin{verbatim}
\let ^^a0=~
\end{verbatim}

Old macro packages need these funny control sequences. We don’t use them in new macros.

\begin{verbatim}
\catcode`@=11
\let \z@=\zo \let \z@skip=\zoskip
\newdimen\p@ \p@=1pt
\toksdef\toks@=0 \let\voidb@x=\voidbox
\chardef\@ne=1 \chardef\tw@=2 \chardef\thr@@=3 \chardef\sixt@@n=16
\mathchardef\@m=1000 \mathchardef\@M=10000 \mathchardef\@MM=20000
\countdef\m@ne=22 \m@ne=-1
\chardef\@cclv=255 \mathchardef\@cclvi=256
\skipdef\skip@=0 \dimendef\dimen@=0 \dimendef\dimen@i=1
\dimendef\dimen@ii=2 \countdef\count@=255
\def\m@th{\mathsurround\z@}
\def\o@lign{\lineskiplimit\z@ \oalign}
\def\n@space{\nulldelimiterspace\z@ \m@th}
\newdimen\p@renwd \p@renwd=8.75pt
\def\alloc@#1#2#3#4#5{\allocator#5{\csstring#2}#3}
\catcode`@=12
\end{verbatim}

We don’t want to read \texttt{opmac.tex} unless \texttt{\input opmac} is specified.
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 BfTeX 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 (number) without the final _par and the second case prints only first sentence from the paragraph (number) using \lipsumdot.

Lu\TeX 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 Lu\TeX 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 _def\macro #1\par{...}. 

Lu\TeX version 1.13 or older:

\def\begmulti \1\par\bgroup\_wipepar\multiskip\penalty0 \1\def\_Ncols\1(\1)
\setbox5=\vbox\bgroup\let\setxhsize=\relax \penalty-99
\divide\hsize by\_colsep
\mullines=0
\def\par{\ifhmode\endgraf\global\advance\mullines by\_prevgraf\fi}
\def\inclusion \bgroup
\ifx\_partokenname\undefined\% Lu\TeX 1.13 or older:
\setbox5=\vbox\bgroup\_wipepar\multiskip\penalty0 \1\def\_Ncols\1(\1)
\setbox5=\vbox\bgroup\let\setxhsize=\relax \penalty-99
\divide\hsize by\_colsep
\mullines=0
\def\par{\ifhmode\endgraf\global\advance\mullines by\_prevgraf\fi}
We set \partokenname to \par in order to keep the name \par in user name space. I.e. a user can say \def\par(paragraph) for example without crash of processing the document. See section 2.2 for more details about the name space concept.

Moreover, we set \partokencontext to one in order to the \par token is inserted not only at empty lines, but also at the end of \vbox, \vtop and \vcenter if horizontal mode is opened here. This differs from default TeX behavior where horizontal mode is closed in these cases without inserting par token.

We set \partokenset to defined value 1 in order to the macro programmer can easily check these settings in OpTEX format by \ifx\partokenset\undefined ... \else ...\fi.

2.39 Lua code embedded to the format

The file optex.lua is loaded into the format in optex.ini as byte-code and initialized by \everyjob, see section 2.1.

The file implements part of the functionality from luatexbase namespace, nowadays defined by L\LaTeX{} kernel. luatexbase deals with modules, allocators, and callback management. Callback management is a nice extension and is actually used in OpTEX. Other functions are defined more or less just to suit luaotfload’s use.

The allocations are declared in subsection 2.39.2, callbacks are implemented in subsection 2.39.3 and handling with colors can be found in the subsection 2.39.5.

2.39.1 General

Define namespace where some OpTEX functions will be added.

Error function used by following functions for critical errors.

For a \chardef’d, \countdef’d, etc., csname return corresponding register number. The responsibility of providing a \XXdef’d name is on the caller.
MD5 hash of given file.

```lua
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
```

### 2.39.2 Allocators

An attribute allocator in Lua that cooperates with normal OpTeX allocator.

```lua
local alloc = _ENV.alloc or {}
_ENV.alloc = alloc

local attributes = {}
function alloc.new_attribute(name)
    local cnt = tex.count"]_attributealloc"] + 1
    if cnt > 65534 then
        tex.error("No room for a new attribute")
    else
        tex.setcount("global", "]_attributealloc", cnt)
        texio.write_nl("log", "]\"..name.."\"="\attribute\"..tostring(cnt))
        attributes[name] = cnt
        return cnt
    end
end
```

Allocator for Lua functions ("pseudoprimitives"). It passes variadic arguments ("...") like "global" to `token.set_lua`.

```lua
local function_table = lua.get_functions_table()
local 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 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 {}
_END 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 = ()
local function callback(name, fn)
    err("direct registering of callbacks is forbidden, use 'callback.add_to_callback'")
end
```

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).
Table that maps standard (and later user) callback names to their types.

```lua
local default_functions = {}

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",
}
function callback.callback_descriptions(name)
    return callback_description[name] or {}
end

local valid_callback_types = {
    exclusive = true,
    simple = true,
    data = true,
    list = true,
    reverselist = true,
}

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

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(...))
    end

    return call_callback(name, ...)
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)
    if #descriptions == 0 then
        -- Delete the list entirely to allow easy checking of "truthiness".
        callback_functions[name] = nil
    end
    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

function reverse_ipairs(t)
    local i, n = #t + 1, 1
    return function()
        i = i - 1
        if i >= n then
            return i, t[i]
        end
        return i, t[i]
    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.
```
elseif cbtype == "exclusive" then
  -- only one function, at least default function is guaranteed by
  -- create_callback
  return functions[1](...)
elseif cbtype == "simple" then
  -- call all functions one after another, no passing of data
  for _, fn in ipairs(functions) do
    fn(...)
  end
  return
elseif cbtype == "data" then
  -- pass data (first argument) from one function to other, while keeping
  -- other arguments
  local data = (...)
  for _, fn in ipairs(functions) do
    data = fn(data, select(2, ...))
  end
  return data
end

-- list and reverselist are like data, but "true" keeps data (head node)
-- unchanged and "false" ends the chain immediately
local iter
if cbtype == "list" then
  iter = ipairs
elseif cbtype == "reverselist" then
  iter = reverse_ipairs
end
local head = (...)
local new_head
local changed = false
for _, fn in iter(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
```

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.

```
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
    changed = true
  end
  return not changed or head
end

if 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, ...)
new_head = call_callback("post_mlist_to_hlist_filter", head, ...)
if new_head == false then
  node.flush_list(head)
return nil
```
elseif new_head ~= true then
head = new_head
end
return head
end)

For preprocessing boxes just before shipout we define custom callback. This is used for coloring based on attributes. There is however a challenge - how to call this callback? We could redefine \shipout and \pdfxform (which both run ship_out procedure internally), but they would lose their primitive meaning – i.e. \immediate wouldn’t work with \pdfxform. The compromise is to require anyone to run \_preshipout⟨destination box number⟩⟨box specification⟩ just before \shipout or \pdfxform if they want to call pre_shipout_filter (and achieve colors and possibly more).

callback.create_callback("pre_shipout_filter", "list")

tex_setbox = tex.setbox
token_scanint = token.scan_int
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)

Compatiblity with \LaTeX through luatexbase namespace. Needed for luaotfload.

\_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.

callback.add_to_callback("input_level_string", function(n)
  if tex.tracingmacros > 3 then
    return "[" .. n .. "] "
  elseif tex.tracingmacros > 2 then
    return ~" .. string.rep(",",n)
  else
    return ""
  end
end, ".tracingmacros")

2.39.4 Management of PDF page resources

Traditionally, pdfTeX allowed managing PDF page resources (graphics states, patterns, shadings, etc.) using a single tokes register, \pdffageresources. 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 LuaTEX and shouldn’t be added!

XObject forms can also use resources, but there are several ways to make LuaTeX 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.get_page_resources(), or the below \TeX alternative for that.

local pdffdict_mt = {
  tostring = function(dict)
local out = {"<<"}
for k, v in pairs(dict) do
    out[#out+1] = fmt("/%s %s", tostring(k), tostring(v))
end
return table.concat(out, "\n")
end,
}
local function pdf_dict(t)
    return setmetatable(t or {}, pdfdict_mt)
end
optex.pdf_dict = pdf_dict

local resource_dict_objects = {}
local page_resources = {}
local function pdf.add_page_resource(type, name, value)
    local resources = page_resources[type]
    if not resources then
        local obj = pdf.reserveobj()
        pdf.setpageresources(fmt("/%s %d 0 R", pdf.get_page_resources(), type, obj))
        resource_dict_objects[type] = obj
        resources = pdf_dict()
        page_resources[type] = resources
    end
    page_resources[type][name] = value
end
local function pdf.get_page_resources()
    return pdf.getpageresources() or ""
end

define_lua_command("_addpageresource", function()
define_lua_command("_pageresources", function()
tex.print(pdf.get_page_resources())
end)

New “pseudo” primitives are introduced. \_addpageresource\{\textcolor{red}{type}\}\{\textcolor{blue}{PDF name}\}\{\textcolor{green}{PDF dict}\} adds more resources of given resource \textcolor{red}{type} to our data structure. \_pageresources expands to the saved \textcolor{red}{type}s and object numbers.

define_lua_command("_addpageresource", function()
define_lua_command("_pageresources", function()
tex.print(pdf.get_page_resources())
end)

We write the objects with resources to the PDF file in the \texttt{finish\_pdffile} callback.

callback.add_to_callback("finish\_pdffile", function()
call for type, dict in pairs(page_resources) do
    local obj = resource_dict_objects[type]
    pdf.immediateobj(obj, tostring(dict))
end

callback.add_to_callback("finish\_pdffile", function()
call for type, dict in pairs(page_resources) do
    local obj = resource_dict_objects[type]
    pdf.immediateobj(obj, tostring(dict))
end

2.39.5 Handling of colors and transparency using attributes

Because Lua\TeX{} 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.

local node_id = node.id
local node_subtype = node.subtype
local glyph_id = node_id("glyph")
local rule_id = node_id("rule")
local glue_id = node_id("glue")
local hlist_id = node_id("hlist")
local vlist_id = node_id("vlist")
local disc_id = node_id("disc")
local whatsit_id = node_id("whatsit")
local pdfliteral_id = node_id("pdf\_literal")
local pdfsave_id = node_id("pdf\_save")
local pdfrestore_id = node_id("pdf\_restore")
local token_getmacro = token.get_macro
local direct = node.direct
local todirect = direct.todirect
local tonode = direct.tonode
local getfield = direct.getfield
local setfield = direct.setfield
local getwhd = direct.getwhd
local getid = direct.getid
local getlist = direct.getlist
local setlist = direct.setlist
local getleader = direct.getleader
local getattribute = direct.get_attribute
local insertbefore = direct.insert_before
local copy = direct.copy
local traverse = direct.traverse
local one_bp = tex.sp("1bp")

The attribute for coloring is allocated in colors.opm

local color_attribute = registernumber(“_colorattr”)
local transp_attribute = registernumber(“_transpattr”)

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.

local pdf_base_literal = direct.new("whatsit", "pdf_literal")
setfield(pdf_base_literal, "mode", 2) -- direct mode
local function pdfliteral(str)
local literal = copy(pdf_base_literal)
setfield(literal, "data", str)
return literal
end
optex.directpdfliteral = pdfliteral

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.

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
  elseif id == rule_id then
    local width, height, depth = getwhd(n)
    return true, true
  end
  if id == rule_id then
    local width, height, depth = getwhd(n)
    return width < 1, true
  end
  return false, false
end

207
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
                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
                    end
                    current_stroke = new
                end
                if newtr ~= current_tr and fill_needed then
                    if newliteral then
                        newliteral = fmt("%s /tr%d gs", newliteral, newtr)
                    else
                        newliteral = fmt("/tr%d gs", newtr)
                    end
                    current_tr = newtr
                end
                if newliteral then
                    head = insert_before(head, n, pdfliteral(newliteral))
                end
            end
        end
    end
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
end
We also hook into \texttt{luaotfload}'s handling of color and transparency. Instead of the default behavior (inserting colorstack \texttt{whatsthis}s) we set our own attribute. On top of that, we take care of transparency resources ourselves.

The hook has to be registered \emph{after} \texttt{luaotfload} is loaded.

\begin{verbatim}
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)
        token_setmacro("_color:"..strattr, color)
        token_setmacro("_color-s:"..strattr, string.upper(color))
    end
    setattribute(todirect(n), color_attribute, attr)
end
optex.set_node_color = set_node_color

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
\end{verbatim}
2.40 Printing documentation

The \printdoc{filename}{space} and \printdoctail{filename}{space} commands are defined after the file doc.opm is loaded 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 \`⟨sequence⟩` in red, for example \`foo`. The user documentation point is the first occurrence of \^`⟨sequence⟩`, for example \^`foo`. There can be more such markups, all of them are hyperlinks to the main documentation point. And main documentation point is a hyperlink to the user documentation point if this point precedes. Finally, the \~`⟨sequence⟩` (for example \~`foo`) are hyperlinks to the user documentation point.

By default, the hyperlink from main documentation point to the user documentation point is active only if it is backward link, i.e. the main documentation point is given later. The reason is that we don’t know if such user documentation point will exist when creating main documentation point and we don’t want broken links. If you are sure that user documentation point will follow then use prefix \fw before \`, for example \fw`foo` is main documentation point where the user documentation point is given later and forward hyperlink is created here.

Control sequences and their page positions of main documentation points and user documentation points are saved to the index.

The listing mode creates all control sequences which are listed in the index as an active link to the main documentation point of such control sequence and prints them in blue. Moreover, active links are control sequences of the type \_foo or .foo although the documentation mentions only \foo. Another text is printed in black.

The listing mode is able to generate external links to another OpTeX-like documentation, if the macros \,⟨csname⟩ and \el:⟨csname⟩ are defined. The second macro should create a hyperlink using \_tmpa where the link name of the ⟨csname⟩ is saved and \_tmpb where the name of the ⟨csname⟩ to be printed is saved (\tmpb can include preceding _ or . unlike \_tmpa). For example, suppose, that we have created optex-doc.eref file by:

TEXINPUTS='.;$TEXMF/{doc,tex}//' optex optex-doc
grep Xindex optex-doc.ref > optex-doc.eref

The .eref file includes only \_Xindex{⟨csname⟩}{} lines from optex-doc.ref file. Then we can use following macros:

\def\Xindex#1\#2{\sdef{,#1\#2}{}\slet{el:\#1}{optexdoclink}}
\def\optexdoclink{%
  \def\extlink{url:optexdocurl\csstring\#cs:\_tmpa}%
  \_ea\_urlactive\_ea[\extlink]{\Cyan}{\csstring\_tmpb}}
All \csname\endcsname, where \csname\endcsname is from \texttt{optex-doc.ref}, have the same meaning: \texttt{optexdoclink} in this example. And \texttt{optexdoclink} creates the external link in \texttt{Cyan} color.

## 2.40.1 Implementation

General decalarations.

```latex
\_codedecl \printdoc{Macros for documentation printing <2022-11-21>} % loaded on demand by \load[doc]
```

Maybe, somebody needs \texttt{\seccc} or \texttt{\secccc}?

```latex
\eodef{\seccc#1\par\noindent\bf#1\par\nobreak\firstnoindent}{\medskip\noindent{\bf#1}\par\nobreak\firstnoindent}
\def{\secccc}{\medskip\noindent $\bullet$}
```

\texttt{\enddocument} can be redefined.

```latex
\_let\enddocument=\_bye
```

A full page of listing causes underfull \texttt{vbox} in output routine. We need to add a small tolerance.

```latex
\pgbottomskip=0pt plus10pt minus2pt
```

The listing mode is implemented here. The \texttt{\maxlines} is maximal lines of code printed in the listing mode. The \texttt{\catcode`\.=11} sets dot as letter in listings (for package documentation where \texttt{\.foo} sequences exist).

```latex
\_newcounter{\_maxlines};\_maxlines=100000
\def{\maxlines}{\_maxlines}
\_public{\maxlines};
```

```latex
\begin{verbatim}
\_edef{\_cod#1\par\_wipeepar}
\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip\_begingroup\_typosize[8/10]\_let\_printverbline=\_printcodeline\_vskip\parskip\_medskip\_ttskip
```
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 \_intlink inside listing mode. But you can define \el: \csname which has precedence and it can create an external link. The \_tmpa includes the name used in the link and \_tmpb is the name to be printed. See \_makecsF above and the example at the beginning of this section.

The lines in the listing mode have a yellow background.

\docfile is currently documented file. \printdoc and \printdoctail macros are defined here.
You can do \verb!\vitt{filename} (\from\to) \verb{filename}! if you need analogical design like in listing mode.

The Index entries are without the trailing backslash in .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 \_afteroad macro.

The \verb{<something>} will be print as \verb{⟨something⟩}.

Main documentation points and hyperlinks to/from it. Main documentation point: \verb{\foo}. User documentation point: \verb{\^\foo}, first occurrence only. The next occurrences are only links to the main documentation point. Link to user documentation point: \verb{~\foo}.
The \fw macro for forward links to user documentation point (given later) is defined here.

\def\fw#1\{\slet{cs:\csstring#1}{}\#1\}
\public\fw;
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.

| \_abovelistskip | 133 |
| \_abovetitle | 128, 130 |
| \activequotes | 192 |
| \addcitelist | 158 |
| \adddcolor | 116 |
| \addextgstate | 115, 147 |
| \additcorr | 107 |
| \addtpageresource | 147, 206 |
| \address | 25, 184–185 |
| \addtabitemx | 152 |
| \addto | 28, 39, 109 |
| \addtomodlist | 79 |
| \addUmathfont | 94, 97 |
| \adef | 17, 28, 39 |
| \adots | 89 |
| \advancepageno | 108, 110 |
| \afterfi | 28, 42 |
| \afteritcorr | 107 |
| \_afterload | 53 |
| \aheadto | 28, 39, 97 |
| \_allocator | 40 |
| \allowbreak | 58 |
| \altquotes | 191–192 |
| \_asciisortingtrue | 179 |
| \_athe | 134 |
| \_authorname | 161–162 |
| \b | 60 |
| \_backgroundbox | 109 |
| \backgroundpic | 145 |
| \_backgroundcolumns | 155 |
| \_banner | 39 |
| \basefilename | 28, 36 |
| \bbchar | 83, 100 |
| \begblock | 14, 27, 135 |
| \begitems | 13–14, 27, 49, 134 |
| \begmulti | 19, 27, 50, 155 |
| \begoutput | 108–109, 125 |
| \begtt | 16–18, 27, 48–49, 109, 136, 138 |
| \_begtt | 136 |
| \_belowliskip | 133 |
| \_belowtitle | 128, 130 |
| \_betweencolumns | 155 |
| \bf | 8–9, 67–68, 76, 83, 100 |
| \bgroup | 38 |
| \bi | 8–9, 67–68, 76, 83, 100 |
| \bib | 20–21, 27, 159 |
| \bibA | 160 |
| \bibB | 160 |
| \bibgl | 160 |
| \bibmark | 157, 160, 162 |
| \bibnum | 158 |
| \bibnum | 121, 157 |
| \_biboptions | 50, 166 |
| \bibp | 157 |
| \bibpart | 21, 50, 157 |
| \bibskip | 160 |
| \bibtextbook | 50, 161 |
| \bibwarning | 161, 165 |
| \big | 88 |
| \Big | 88 |
| \_addcolor | 116 |
| \addextgstate | 115, 147 |
| \_additcorr | 107 |
| \_addtpageresource | 147, 206 |
| \address | 25, 184–185 |
| \_addtabitemx | 152 |
| \addto | 28, 39, 109 |
| \_addtomodlist | 79 |
| \addUmathfont | 94, 97 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \bigbreak | 58 |
| \bigskip | 58 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |
| \_bibA | 160 |
| \_bibB | 160 |
| \_bibgl | 160 |
| \_bibwarning | 161, 165 |
| \Big | 88 |