OpTEX

Format Based on Plain TEX and OPmac

Version 1.08

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.
2.2.6 The implementation of the namespaces ............................................. 34
2.3 pdfTeX initialization ................................................................. 35
2.4 Basic macros ............................................................................... 37
2.5 Allocators for \TeX registers ......................................................... 39
2.6 If-macros, loops, is-macros .......................................................... 41
  2.6.1 Classical \newif ......................................................................... 41
  2.6.2 Loops ..................................................................................... 42
  2.6.3 Is-macros ............................................................................... 43
2.7 Setting parameters ......................................................................... 45
  2.7.1 Primitive registers ................................................................... 45
  2.7.2 Plain \TeX registers ................................................................. 46
  2.7.3 Different settings than in plain \TeX .............................................. 46
  2.7.4 Op\TeX parameters ................................................................. 47
2.8 More Op\TeX macros ...................................................................... 51
  2.9 Using key=value format in parameters .......................................... 54
2.10 Plain \TeX macros ......................................................................... 55
2.11 Preloaded fonts for text mode ....................................................... 60
  2.12 Using \font primitive directly ...................................................... 60
  2.12.1 The \setfontsize macro ........................................................ 61
  2.12.2 The \fontlet declarator ......................................................... 62
  2.12.3 Optical sizes ........................................................................... 62
  2.12.4 Font rendering ........................................................................ 62
  2.12.5 Implementation of resizing ...................................................... 62
2.13 The Font Selection System ............................................................ 64
  2.13.1 Terminology ............................................................................ 64
  2.13.2 Font families, selecting fonts ................................................... 65
  2.13.3 Math Fonts ............................................................................. 65
  2.13.4 Declaring font commands ...................................................... 65
  2.13.5 The \fontdef declarator in detail .......................................... 66
  2.13.6 The \famvardef declarator .................................................... 66
  2.13.7 The \tt variant selector .......................................................... 67
  2.13.8 Font commands defined by \def ................................................. 67
  2.13.9 Modifying font features ........................................................... 68
  2.13.10 Special font modifiers ........................................................... 68
  2.13.11 How to create the font family file ........................................... 68
  2.13.12 How to write the font family file with optical sizes ................. 71
  2.13.13 How to register the font family in the Font Selection System ... 73
  2.13.14 Implementation of the Font Selection System ....................... 74
2.14 Preloaded fonts for math mode ..................................................... 80
2.15 Math macros ................................................................................. 82
2.16 Unicode-math fonts ...................................................................... 91
  2.16.1 Unicode-math macros preloaded in the format ......................... 92
  2.16.2 Macros and codes set when \loadmath is processed firstly .......... 95
  2.16.3 More Unicode-math examples ................................................. 101
  2.16.4 Printing all Unicode math slots in used math font ................. 101
2.17 Scaling fonts in document (high-level macros) ................................ 102
2.18 Output routine ............................................................................ 105
2.19 Margins ....................................................................................... 108
2.20 Colors ......................................................................................... 109
  2.20.1 Basic concept ......................................................................... 109
  2.20.2 Color mixing .......................................................................... 109
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.20.3</td>
<td>Implementation</td>
<td>110</td>
</tr>
<tr>
<td>2.21</td>
<td>The <code>.ref</code> file</td>
<td>114</td>
</tr>
<tr>
<td>2.22</td>
<td>References</td>
<td>116</td>
</tr>
<tr>
<td>2.23</td>
<td>Hyperlinks</td>
<td>118</td>
</tr>
<tr>
<td>2.24</td>
<td>Making table of contents</td>
<td>120</td>
</tr>
<tr>
<td>2.25</td>
<td>PDF outlines</td>
<td>122</td>
</tr>
<tr>
<td>2.25.1</td>
<td>Nesting PDF outlines</td>
<td>122</td>
</tr>
<tr>
<td>2.25.2</td>
<td>Strings in PDF outlines</td>
<td>123</td>
</tr>
<tr>
<td>2.26</td>
<td>Chapters, sections, subsections</td>
<td>125</td>
</tr>
<tr>
<td>2.27</td>
<td>Lists, items</td>
<td>130</td>
</tr>
<tr>
<td>2.28</td>
<td>Verbatim, listings</td>
<td>132</td>
</tr>
<tr>
<td>2.28.1</td>
<td>Inline and “display” verbatim</td>
<td>132</td>
</tr>
<tr>
<td>2.28.2</td>
<td>Listings with syntax highlighting</td>
<td>136</td>
</tr>
<tr>
<td>2.29</td>
<td>Graphics</td>
<td>139</td>
</tr>
<tr>
<td>2.30</td>
<td>The <code>\table</code> macro, tables and rules</td>
<td>145</td>
</tr>
<tr>
<td>2.30.1</td>
<td>The boundary declarator :</td>
<td>145</td>
</tr>
<tr>
<td>2.30.2</td>
<td>Usage of the <code>\tabskip</code> primitive</td>
<td>145</td>
</tr>
<tr>
<td>2.30.3</td>
<td>Tables to given width</td>
<td>146</td>
</tr>
<tr>
<td>2.30.4</td>
<td><code>\eqbox</code> : boxes with equal width across the whole document</td>
<td>146</td>
</tr>
<tr>
<td>2.30.5</td>
<td>Implementation of the <code>\table</code> macro and friends</td>
<td>147</td>
</tr>
<tr>
<td>2.31</td>
<td>Balanced multi-columns</td>
<td>152</td>
</tr>
<tr>
<td>2.32</td>
<td>Citations, bibliography</td>
<td>154</td>
</tr>
<tr>
<td>2.32.1</td>
<td>Macros for citations and bibliography preloaded in the format</td>
<td>154</td>
</tr>
<tr>
<td>2.32.2</td>
<td>The <code>\usebib</code> command</td>
<td>157</td>
</tr>
<tr>
<td>2.32.3</td>
<td>Notes for bib-style writers</td>
<td>157</td>
</tr>
<tr>
<td>2.32.4</td>
<td>The <code>usebib.opm</code> macro file loaded when <code>\usebib</code> is used</td>
<td>159</td>
</tr>
<tr>
<td>2.32.5</td>
<td>Usage of the <code>bib-iso690</code> style</td>
<td>161</td>
</tr>
<tr>
<td>2.32.6</td>
<td>Implementation of the <code>bib-iso690</code> style</td>
<td>168</td>
</tr>
<tr>
<td>2.33</td>
<td>Sorting and making Index</td>
<td>172</td>
</tr>
<tr>
<td>2.34</td>
<td>Footnotes and marginal notes</td>
<td>179</td>
</tr>
<tr>
<td>2.35</td>
<td>Styles</td>
<td>181</td>
</tr>
<tr>
<td>2.35.1</td>
<td><code>\report</code> and <code>\letter</code> styles</td>
<td>181</td>
</tr>
<tr>
<td>2.35.2</td>
<td><code>\slides</code> style for presentations</td>
<td>182</td>
</tr>
<tr>
<td>2.36</td>
<td>Logos</td>
<td>185</td>
</tr>
<tr>
<td>2.37</td>
<td>Multilingual support</td>
<td>186</td>
</tr>
<tr>
<td>2.37.1</td>
<td>Lowercase, uppercase codes</td>
<td>186</td>
</tr>
<tr>
<td>2.37.2</td>
<td>Multilingual phrases and quotation marks</td>
<td>187</td>
</tr>
<tr>
<td>2.37.3</td>
<td>Languages declaration</td>
<td>189</td>
</tr>
<tr>
<td>2.37.4</td>
<td>Data for various languages</td>
<td>192</td>
</tr>
<tr>
<td>2.38</td>
<td>Other macros</td>
<td>193</td>
</tr>
<tr>
<td>2.39</td>
<td>Lua code embedded to the format</td>
<td>195</td>
</tr>
<tr>
<td>2.39.1</td>
<td>General</td>
<td>196</td>
</tr>
<tr>
<td>2.39.2</td>
<td>Allocators</td>
<td>196</td>
</tr>
<tr>
<td>2.39.3</td>
<td>Callbacks</td>
<td>197</td>
</tr>
<tr>
<td>2.39.4</td>
<td>Management of PDF page resources</td>
<td>202</td>
</tr>
<tr>
<td>2.39.5</td>
<td>Handling of colors and transparency using attributes</td>
<td>203</td>
</tr>
<tr>
<td>2.40</td>
<td>Printing documentation</td>
<td>206</td>
</tr>
<tr>
<td>2.40.1</td>
<td>Implementation</td>
<td>207</td>
</tr>
</tbody>
</table>

Index 211
Chapter 1
User documentation

1.1 Starting with OpTEX

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

```
optex document
```

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

If there is no `optex` command, see more information about installation OpTEX at [http://petr.olsak.net/optex](http://petr.olsak.net/optex).

A minimal document should be

```
\fontfam[LMfonts]
Hello World! \bye
```

The first line `\fontfam[LMfonts]` tells that Unicode Latin Modern fonts (derived from Computer Modern) are used. If you omit this line then preloaded Latin Modern fonts are used but preloaded fonts cannot be in Unicode\(^1\). So the sentence `Hello World` will be OK without the first line, but you cannot print such sentence in other languages (for example `Ahoj světe!`) where Unicode fonts are needed because the characters like è are not mapped correctly in preloaded fonts.

A somewhat larger example with common settings should be:

```
\fontfam[Termes] % selecting Unicode font family Termes (section 1.3.1)
\typosize[11/13] % setting default font size and baselineskip (sec. 1.3.2)
\margins/1 a4 (1,1,1,1)in % setting A4 paper, 1 in margins (section 1.2.1)
\cslang % Czech hyphenation patterns (section 1.7.1)
```

```
Tady je zkušební textík v českém jazyce.
\bye
```

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

1.2 Page layout

1.2.1 Setting the margins

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

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

example:

```
\margins/1 a4 (2.5,2.5,2,2)cm
```

Parameters are:

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

---

\(^1\) This is a technical limitation of LuaTeX for fonts downloaded in formats: only 8bit fonts can be preloaded.
Each of the parameters \langle left\rangle, \langle right\rangle, \langle top\rangle, \langle bot\rangle can be empty. If both \langle left\rangle and \langle right\rangle are nonempty then \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:

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

If \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.

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

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

The \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:

\margins/1 (100,200) (7,7,7,7)mm
declarres the paper 100\times200mm with all four margins 7mm. The spaces before and after \langle fmt\rangle parameter are necessary.

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:

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

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

1.2.2 Concept of the default page

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

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

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

\headline = {} \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
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{upbackground} 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\footnote{You can declare \texttt{fnotenumglobal} if you want footnotes numbered in whole document from one or \texttt{fnotenumpages} if you want footnotes numbered at each page from one. Default setting is \texttt{fnotenumchapters}.}. The \texttt{⟨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 \texttt{⟨text⟩} to the footnote. The \texttt{⟨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{fnotetexts} have to be written after the box as the number of \texttt{fnotemarks} inserted inside the box. Example:

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

The marginal note can be printed by the \texttt{mnote{⟨text⟩}} macro. The \texttt{⟨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 \texttt{⟨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 \texttt{⟨dimen⟩} to each \texttt{mnote} manually in final printing in order to margin notes do not overlap. The positive value of \texttt{⟨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 \texttt{(Family-name)} is case insensitive and spaces are ignored in it. For example, \texttt{\fontfam[LM Fonts]} is equal to \texttt{\fontfam[LMfonts]} and it is equal to \texttt{\fontfam[lmfonts]}. Several aliases are prepared, thus \texttt{\fontfam[Latin Modern]} can be used for loading Latin Modern family too.

If you write \texttt{\fontfam[?]} then all font families registered in Op\TeX are listed on the terminal and in the log file. If you write \texttt{\fontfam[catalog]} then a catalog of all fonts registered in

\footnote{You can declare \texttt{fnotenumglobal} if you want footnotes numbered in whole document from one or \texttt{fnotenumpages} if you want footnotes numbered at each page from one. Default setting is \texttt{fnotenumchapters}.}
Op\TeX and available in your \TeX system is printed. The instructions on how to register your own font family are appended in the catalog.

If the family is loaded then font modifiers applicable in such font family are listed on the terminal: (\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 \sl is inserted automatically, if needed.

Example:

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

More about the Op\TeX 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 \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 \ptunit which is 1pt by default. You can change the unit by the command \ptunit=⟨something-else⟩, for instance \ptunit=1mm enlarges all font sizes declared by \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] \ldots ⟨text of paragraph⟩ \ldots \par}.

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

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

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

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

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

More information about resizing of fonts is documented in the section 2.12.1.

1.3.3 Typesetting math

See the additional document \texttt{Typesetting Math with \texttt{\textbackslash Opti\textbackslash Ti\textbackslash X}} for more details about this issue.

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

Hundreds math symbols and operators like in AMSTEX are accessible. For example \texttt{\alpha}, \texttt{\\geq}, \texttt{\\sum\sum}, \texttt{\sphericalangle \angle}, \texttt{\bumpeq \approx}. See AMSTEX manual or \texttt{Typesetting Math with \texttt{\textbackslash Opti\textbackslash Ti\textbackslash X}} for complete list of math symbols.

The following math alphabets are available:

\begin{verbatim}
\mit % mathematical variables abc–xyz, ABC–XYZ
\it % text italics abc–xyz, ABC–XYZ
\rm % text roman abc–xyz, ABC–XYZ
\cal % normal calligraphics ABC–XYZ
\script % script \mathcal{ABC}–XYZ
\frak % fracture abc–\mathfrak{xyz}, \mathfrak{ABC}–\mathfrak{XYZ}
\bbchar % double stroked letters ABC–XYZ
\bf % sans serif bold abc–xyz, ABC–XYZ
\bi % sans serif bold slanted abc–xyz, ABC–XYZ
\end{verbatim}

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

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

1.4 Typical elements of the document

1.4.1 Chapters and sections

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

\tit Document title \text{(end of line)}
\chap Chapter title \text{(end of line)}
\sec Section title \text{(end of line)}
\secc Subsection title \text{(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 \nl if you want to have corresponding lines in the source and the output.

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

1.4.2 Another numbered objects

Apart from chapters, sections, and subsections, there are another automatically numbered objects: equations, captions for tables and figures. The user can declare more numbered objects. If the user writes the \eqmark as the last element of the display mode then this equation is numbered. The equation number is printed in brackets. This number is reset in each section by default.

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

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

Another automatically numbered object is a caption which is tagged by \caption/t for tables and \caption/f for figures. The caption text follows. The \cskip can be used between \caption text and the real object (table or figure). You can use two orders: \text{(caption)}\cskip \text{(object)} or \text{(object)}\cskip \text{(caption)}. 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.

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

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

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

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

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

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

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

Three independent numbers are used in this example. One for Theorems and Corollaries second for Definitions and third for Examples. The user can write \theorem Let $M$ 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...
1.4.3 References

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

\sec[beatle] About Beatles
\noindent\hfil\table{rl}{...} % the table
\cskip
\caption/t [comp-depend] The dependency of the comp-dependency on the age.

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

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

The text printed by \ref or \pgref can be given explicitly by \ref[\(label\)]{\(text\)} or \pgref[\(label\)]{\(text\)}. If the \(text\) includes the \& character, it is replaced by implicitly printed text. Example: see \ref[lab]{section-\&} prints the same as see section~\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 \label[\(label\)] before the \theorem, \definition etc. (macros defined with \numberedpar) if you want to reference these numbered objects. You can’t use \theorem[\(label\)] because the optional parameter is reserved to another purpose here.

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

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

1.4.4 Hyperlinks, outlines

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

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

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

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

You can use another marking of active links: by frames which are visible in the PDF viewer but invisible when the document is printed. The way to do it is to define the macros \_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\{\color-fnt\} \{\color-fnf\} where footnote marks in the text have \{\color-fnt\} and the same footnote marks in footnotes have \{\color-fnf\}. You can define relevant borders \_fntborder and \_fnfborder analogically as \_pgborder (for example).

There are “low level” commands to create the links. You can specify the destination of the internal link by \dest\{\langle\text\rangle\}. The active text linked to the \dest can be created by \ilink\{\langle\text\rangle\}\{\langle\text\rangle\}. The \{\langle\text\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 argument. The other special characters ~, ^, &, can be written as single character.

You can insert the \| command in the \url argument as a potential breakpoint.

If the linked text have to be different than the URL, you can use \ulink\{\langle\url\rangle\}\{\langle\text\rangle\} macro. For example: \ulink[http://petr.olsak.net/optex]{\OpTeX/ page} outputs to the text \OpTeX/ page. The characters %, \, #, { and } must be escaped in the \url parameter.

The PDF format provides outlines which are notes placed in the special frame of the PDF viewer. These notes can be managed as a structured and hyperlinked table of contents of the document. The command \outlines\{\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), ...
\textit{\style A} % items of type A), B), C), ...
\textit{\style x} % small rectangle
\textit{\style X} % big rectangle

For example:

\begitens
* First idea
* Second idea in subitems:
  \begitens \style i
  * First sub-idea
  * Second sub-idea
  * Last sub-idea
  \enditens
* Finito
\enditens

produces:

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

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

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

You can say \begitens \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{\{\text\}}{\{\text\}} provides similar \{\text\} of tables as in \LaTeX: you can use letters l, r, c, each letter declares one column (aligned to left, right, center, respectively). These letters can be combined by the | character (vertical line). Example

\table{||l|c|r||}{
\begin\tskip2pt
Month & commodity & price \\
January & notebook & \$ 700 \\
February & skateboard & \$ 100 \\
July & yacht & k\$ 170 \\
\crl}

generates the result:

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

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

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

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

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

- \cr ... the end of the row with a horizontal line after it.
- \crll ... the end of the row with a double horizontal line after it.
- \crii ... like \cr but the horizontal line doesn’t intersect the vertical double lines.
- \criii ... like \crll but horizontal line is doubled.
- \crlp{⟨list⟩} ... like \crii 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
\thisetable={} % 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
\vkern={\ipt} % space between lines in double vertical line
\hkern={\ipt} % space between lines in double horizontal line
\tabskip=0pt % space between columns
\tabskipl=0pt \tabskipr=0pt % space before first and after last column

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

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

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

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

The \texttt{\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:

\begin{verbatim}
\thistable{\tabstrut=\{\vrule height 20pt depth10pt width0pt\}
    \baselineskip=20pt \tablinespace=0pt \rulewidth=.8pt}
    \table{18\{c\}|}\{\crlp\{3-8\}
        \mspan2\{c\}|\{\vspan\{\text\}\} & \mspan3\{c\}|\{\text\}\{\vspan\{\text\}\} & \mspan3\{c\}|\{\text\}\{\vspan\{\text\}\} \{\crlp\{3-8\}
        \mspan2\{c\}|\{\text\}\{\vspan\{\text\}\} & \mspan2\{c\}|\{\text\}\{\vspan\{\text\}\} \{\crlp\{3-8\}
        \vspan\{\text\}\{\vspan\{\text\}\} & \mspan3\{c\}|\{\text\}\{\vspan\{\text\}\} \{\crlp\{2,6-8\}
        \vspan\{\text\}\{\vspan\{\text\}\} & \mspan3\{c\}|\{\text\}\{\vspan\{\text\}\} \{\crlp\{2-8\}
        \vspan\{\text\}\{\vspan\{\text\}\} & \mspan6\{c\}|\{\text\}\{\vspan\{\text\}\} \{\crlp\{1\}
        \vspan\{\text\}\{\vspan\{\text\}\} \{\crlp\{2,4-7\}
        \vspan\{\text\}\{\vspan\{\text\}\} & \mspan4\{c\}|\{\text\}\{\vspan\{\text\}\} \{\crlp\}
    }\end{verbatim}

<table>
<thead>
<tr>
<th></th>
<th>Singular</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neuter</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Masculine</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Feminine</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

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

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

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

1.4.7 Verbatim

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

If the numerical register \texttt{\ttline} is set to the non-negative value then display verbatim will number the lines. The first line has the number \texttt{\ttline+1} and when the verbatim ends then the \texttt{\ttline} value is equal to the number of the last line printed. Next \texttt{\begtt...\endtt} environment will follow the line numbering. \TeX sets \texttt{\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 \fnotes, because these macros are specially defined in OpTEX). You can use more robust command \code{(text)} in problematic situations, but you have to escape the following characters in the (text): \, #, %, braces (if the braces are unmatched in the (text)), and space or ^ (if there are more than one subsequent spaces or ^ in the (text)). Examples:

\code{\text, \%\#} ... prints \text, %#
\code{@(\text} ... prints @{\text 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 \verbinput command. Examples:

\verbinput (12-42) program.c % listing from program.c, only lines 12-42
\verbinput (-60) program.c % print from begin to the line 60
\verbinput (61-) program.c % from line 61 to the end
\verbinput (-) program.c % whole file is printed
\verbinput (70+10) program.c % from line 70, only 10 lines printed
\verbinput (+10) program.c % from the last line read, print 10 lines
\verbinput (-5+7) program.c % from the last line read, skip 5, print 7
\verbinput (+) program.c % from the last line read to the end
You can insert additional commands for \verbinput before the first opening bracket. They are processed in the local group. For example, \verbinput \hsize=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<0 then no line numbers are printed.

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

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

If the code is read by \verbinput and there are comment lines prefixed by two characters then you can set them by \commentchars{first}{second}. Such comments are fully interpreted by \TeX (i.e. not verbatim). Section 2.28.1 (page 135) says more about this feature.

### 1.5 Autogenerated lists

#### 1.5.1 Table of contents

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

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

\nonum\notoc\sec Table of Contents

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

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

\% default value of \mylogo macro used in text and in the titles:
\def\mylogo{\leavevmode\hbox{{\Red\it My}{\setfontsize{mag1.5}\rm Lo}Go}}

\% another variants:
\regmacro {\def\mylogo{\hbox{\Red My\Black LoGo}}}% used in TOC
{\def\mylogo{\hbox{{\it My}/\LoGo}}}% used in running heads
{\def\mylogo{MyLoGo}}% used in PDF outlines

#### 1.5.2 Making the index

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

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

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

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

or:

Now we'll deal with the \iid resistor.

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

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

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

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

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

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

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

\ii linear/space,vector/space,@

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

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

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

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

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

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

\sec{Index}
\begmulti 3 \makeindex \endmulti

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

The \ii chiquadrat $\chi$-quadrat method is ...

If the \ii relax `\relax` command is used then \TeX/ is relaxing.

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

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

1.5.3 \texttt{BibT\TeX}Xing

The command \cite{\{label\}} (or \cite{\{label-1\},\{label-2\},...,\{label-n\}}) creates the citation in the form \cite{\{label\}} (or \cite{\{label-1\}, \{label-2\}, ..., \{label-n\}}). If \texttt{shortcitations} is declared at the beginning of the document then continuous sequences of numbers are re-printed like this: \cite{\{label\}} \cite{\{label-1\}} \cite{\{label-2\}}. If \texttt{sortcitations} is declared then numbers generated by one \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 \cite{\{label\}} or \cite{\{label\}, \{label-1\}, \{label-2\}, ..., \{label-n\}}. If \texttt{hyperlinks} is used then \{text\} is linked to the references list.

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

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

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

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

Note that another two possibilities documented in OPmac (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\X.

References created manually using \texttt{\bib{\{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 \{marks\} used by \cite command. To do it you must use long form of the \bib command in the format \bib\{\{label\}\} = \{\{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 (\{style\}) \{bib-base\} % sorted by \cite-order (c=cite),
\usebib/s (\{style\}) \{bib-base\} % sorted by style (s=style).
\end{verbatim}

% example:
% \nocite[*] \usebib/s (simple) op-biblist % prints all from op-biblist.bib

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

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

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

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

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

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

## 1.6 Graphics

### 1.6.1 Colors, transparency

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

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

The color selectors work locally in groups like font selectors.

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

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

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

---
\(^5\) Printed output is more equal to the monitor preview especially if you are using ICC profile for your printer.
The linear combination is done in CMYK subtractive color space by default (RGB colors used in \colordef argument are converted first). If the resulting component is greater than 1 then it is truncated to 1. If a convex linear combination (as in the last example above) is used then it emulates color behavior on a painter’s palette. You can use \rgbcolordef instead of \colordef if you want to mix colors in the additive RGB color space. If onlyrgb is set then \colordef works like \rgbcolordef.

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

The \coloron macro can be defined as follows:

\def\coloron#1#2#3{\setbox0=\hbox{#2#3}\leavevmode \rlap{#1\strut} \vrule width\wd0} \box0

\coloron\Yellow\Brown{Brown text on yellow background}

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

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

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

Inkscape\textsuperscript{6} is able to save a picture to PDF and labels of the picture to another file\textsuperscript{7}. This second file should be read by \TeX to print labels in the same font as document font. \opTeX supports this feature by \inkinspic {⟨filename⟩.pdf} command. It reads and displays both: PDF image and labels generated by Inkscape.

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

1.6.3 PDF transformations

All typesetting elements are transformed by linear transformation given by the current transformation matrix. The \pdfsetmatrix \{⟨a⟩ ⟨b⟩ ⟨c⟩ ⟨d⟩\} command makes the internal

---

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

\textsuperscript{7} Chose “Omit text in PDF and create \LaTeX file” option.
multiplication with the current matrix so linear transformations can be composed. One linear transformation given by the \texttt{pdfsetmatrix} above transforms the vector \([0, 1]\) to \([⟨a⟩, ⟨b⟩]\) and \([1, 0]\) to \([⟨c⟩, ⟨d⟩]\). The stack-oriented commands \texttt{pdfsave} and \texttt{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 \texttt{pdfrestore} is processed. Due to this fact the \texttt{pdfsave\llap{\text{transformed text}}\pdfrestore} or something similar is recommended.

\OpTeX{} provides two special transformation macros \texttt{pdfscale} and \texttt{pdfrotate}:

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

These macros simply call the properly \texttt{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: \texttt{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: \texttt{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: \texttt{pdfsave}\ pdfrotate\{-15.3\}\ pdfsetmatrix\{2\ 0\ 1.5\ 2\}\ rlap\{text3\}\ %
\texttt{pdfrestore}\ % first slanted, then rotated by 15.3 degrees right

This gives the following result. First second third:

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

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

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

1.6.4 Ovals, circles

The \texttt{inoval\{⟨text⟩\}} creates a box like this: text. Multiline text can be put in an oval by the command \texttt{inoval\{vbox\{⟨text⟩\}}}.

Local settings can be set by \texttt{inoval\{⟨settings⟩\}\{⟨text⟩\}} or you can re-declare global settings by \texttt{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
 lwidth=0.5bp % line width in the border
 shadow=N % use a shadow effect
 overlapmargins=N % ignore margins by surrounding text
 hhkern=Opt \ vvkern=Opt \ % left-right margin, top-bottom margin
\end{verbatim}

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

The \text{\incircle[⟨ratio=1.8⟩]{⟨text⟩}} creates a box like this (text). The \text{⟨ratio⟩} parameter means width/height. The usage is analogous like for oval. The default parameters are

\text{\circleparams={⟨ratio=1 \fcolor=Yellow \lcolor=Red \lwidth=0.5bp \shadow=N \ignoremargins=N \hhkern=2pt \vvkern=2pt}}}.

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

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

1.6.5 Putting images and texts wherever

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

The \text{\putpic ⟨x⟩ ⟨y⟩ ⟨width⟩ ⟨height⟩ {⟨image-file⟩}} puts an image given by \text{⟨image-file⟩} (including extension) of given \text{⟨width⟩} and \text{⟨height⟩} at given position (its left-bottom corner). You can write \text{\nospec} instead \text{⟨width⟩} or \text{⟨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 \text{\lang{id}lang} command in your document. For example \text{\delang} for German, \text{\pllang} for Czech, \text{\pllang} for Polish. The \text{⟨lang-id⟩} is a shortcut of the language (mostly from ISO 639-1). You can list all available languages including their \text{⟨lang-id⟩}’s by the \text{\langlist} macro. It prints now:

en(USEnglish) enmax(UKEnglishmax) engb(UKEnglish) be(Belarusian) bg(Bulgarian) ca(Catalan) hr(Croatian) cs(Czech) da(Danish) nl(Dutch) et(Estonian) fi(Finnish) fr(French) de(aGerman) deo(oldGerman) gsw(swiss-German) el(pGreek) greek(ancientGreek) hu(Hungarian) is(Icelandic) it(Italian) ia(Interlingua) ja(Japanese) ko(Korean) la(Latin) lat(latin) latvian(Latvian) lt(Lithuanian) mk(Macedonian) pl(Polish) pt(Portuguese) ro(Romanian) ru(Russian) sr(Serbian) sv(Swedish) sv(Swedish) uk(Ukrainian) cy(Welsh) af(Afrikaans) hy(Armenian) as(Assamese) bn(Bengali) cop(Coptic) cu(churchslavonic) eo(Esperanto) eth(Ethiopic) fur(Friulian) gl(Galician) ka(Georgian) gu(Gujarati) hi(Hindi) id(Indonesian) ia(Interlingua) kn(Kannada) km(Khmer) mi(Maori) mk(Macedonian) mn(Mongolian) mr(Marathi) ne(Nepali) no(Norwegian) oc(Occitan) or(Orish) pa(Pali) pl(Pashami) pm(Piedmontese) zh(Pinyin) sa(Sanskrit) te(Telugu) th(Thai) tr(Turkish) tk(Turkmen) hu(UpperSorbian)

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

For compatibility with e-plain, there are macros \text{\ehyph}, \text{\ehyph}, \text{\ehyph} which are equivalent to \text{\enlang}, \text{\cslang} and \text{\kslang}.

You can switch between language patterns by \text{\lang{id}lang} commands mentioned above. Default is \text{\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 \text{\lang{id}lang}. 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 \edef or \_langw commands as shown in section 2.37.2.

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

If you declare \langle iso-code \rangle quotes, then the control sequences \" and \' should be used like this: \"{(quoted text)}" or \'{quoted text}' (note that the terminating character is the same but it isn’t escaped). This prints language-dependent normal or alternative quotes around \{(quoted text)\}. The language is specified by \langle iso-code \rangle. OpTEX declares quotes only for Czech, German, Spanish, French, Greek, Italian, Polish, Russian, Slovak and English (\csquotes, \dequotes, . . ., \enquotes). You can simply define your own quotes as shown in section 2.37.2. The \" is used for quotes visually more similar to the ” character which can be primary quotes or secondary quotes depending on the language rules. Maybe you want to alternate the meaning of these two types of quotes. Use \langle isocode \rangle quotes\altquotes in such case.

1.7.2 Pre-defined styles

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

The \report declaration is intended to create reports. It sets default font size to 11pt and \parindent (paragraph indentation) to 1.2em. The \tit macro uses smaller font because we assume that “chapter level” will be not used in reports. The first page has no page number, but the next pages are numbered (from number 2). Footnotes are numbered from one in the whole document. The macro \author \langle authors \rangle \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 examples. The \letter style sets default font size to 11pt and \parindent to 0pt. It sets half-line space between paragraphs. The page numbers are not printed. The \subject macro can be used, it prints the word “Subject:” or “Věc” (or something else depending on current language) in bold. Moreover, the \address macro can be used when \letter is declared. The usage of the \address macro looks like:

\begin{verbatim}
\address
  \langle first line of address \rangle
  \langle second line of address \rangle
  \langle etc. \rangle
  \langle empty line \rangle
\end{verbatim}

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

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

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

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

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

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

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

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

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

1.7.4 Lorem ipsum dolor sit

A designer needs to concentrate on the design of the output and maybe he/she needs material for testing macros. There is the possibility to generate a neutral text for such experiments. Use \texttt{\lorem[⟨number⟩]} or \texttt{\lorem[⟨from⟩-⟨to⟩]}\footnote{If the dot follows the argument before closing ] (for example \texttt{\lorem[3.]} then only first sentence from given paragraph is printed.}. 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 \texttt{\lipsum} macro is equivalent to \texttt{\lorem}. Example: \texttt{\lipsum[1-150]} prints all prepared paragraphs.

1.7.5 Logos

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

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

1.7.6 The last page

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

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

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

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

The command `\useOpTeX` (or `\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:

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

1.8 Summary

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

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

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

If you are writing a document or macros specific for the document, then use simply user namespace (\verb|\foo|). If you are writing more general macros, then use private namespace (\verb|\_foo|), but you should declare your own namespace by \verb|\_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.

\verb|\addto \macro{(text)}| adds \texttt{(text)} at the end of \texttt{\macro} body.
\verb|\def {char}{(body)}| defines \texttt{(char)} active character with meaning \texttt{(body)}.
\verb|\afterfi {(text)}| expands to \verb|\fi{(text)}|.
\verb|\bp {(dimen expression)}| expands \TeX{} dimension to decimal number in \verb|\bp| without unit.
\verb|\_encodedcl \sequence\{\info\}| is used at beginning of macro files.
\verb|\colordef \macro \{\mix of colors\}| declares \texttt{\macro} as color switch.
\verb|\cs \{\string\}| expands \texttt{\string}.
\verb|\_doc \_cod| encloses documentation text in the macro code.
\verb|\eoldef \macro #1{(body)}| defines \texttt{\macro} with parameter separated to end of line.
\verb|\_endcode| closes the part of macro code in macro files.
\verb|\_endnamespace| closes name space declared by \verb|\_namespace|.
\verb|\_bbox \{\label\}\{\text\}| creates \verb|\bbox\{\text\}| with common width across whole document.
\verb|\expr \{\expression\}| expands to result of the \texttt{\expression} with decimal numbers.
\verb|\fontdef \f \{\font spec\}| declares \verb|\f| as font switch.
\verb|\fontlet \fa=\fb \{sizespec\}| declares \verb|\fa| as the same font switch like \verb|\fb| at given \texttt{\{sizespec\}}.
\verb|\foreach \list\do {\parameters}\{\what\}| is expandable loop over \texttt{\list}.
\verb|\foreachdef \macro \{\parameters\}\{\what\}| declares expandable \texttt{\macro} as loop over \texttt{\list}.
\verb|\forform \from\{\to\}\do \{\what\}| is expandable loop with numeric variable.
\verb|\incr \{\counter\}| increases and \verb|\decr \{\counter\}| decreases \texttt{\counter} by one globally.
\verb|\ignoreit \{one\}, \ignoresecond \{one\}\{two\} ignores given parameter.
\verb|\expandafter \ignoreopt \theta\{\dimen\}| expands to decimal number \texttt{\dimen} without \verb|pt|.
\verb|\isempty, \listokempty, \isequal, \ismacro, \isdefined, \isinlist \isfile, \isfont| do various tests. Example: \verb|\isinlist\list\{\text\}|\texttt{\iftrue} does \texttt{\iftrue} if \texttt{\text} is in \texttt{\list}.
\verb|\isnextchar \char\{\text1\}\{\text2\}| performs \texttt{\text1} if next character is \texttt{\char}, else \texttt{\text2}.
\verb|\kv \{\key\}| expands to value when key-value parameters are used.
\verb|\loop \{\repeat\}| is classical Plain \TeX{} loop.
\verb|\mathstyles \{\math list\}| enables to create macros dependent on current math style.
\verb|\_namespace \{\pk\}| declares name space used by package writers.
\verb|\newcount, \newdimen| etc. are classical Plain \TeX{} allocators.
\verb|\newif \iffoo| declares boolean \verb|\iffoo| as in Plain \TeX{}.
\verb|\_newifi \_iffoo| declares boolean \verb|\_iffoo|.
\verb|\opinput \{\filename\}| reads file like \verb|\input| but with standard catcodes.
\verb|\optdef \macro \{\opt-default\}\{\parameters\}\{\body\}| defines \texttt{\macro} with \texttt{\{opt-parameter\}}.
\verb|\opwarning \{\text\}| prints \texttt{\text} to the terminal and .log file as warning.
\verb|\private \sequence\{\sequence\}| \texttt{\sequence} ... ; declares \texttt{\sequence}s for private name space.
\verb|\public \sequence\{\sequence\}| \texttt{\sequence} ... ; declares \texttt{\sequence}s for public name space.
\readkv \macro reads parameters from \macro in key-value format.
\replstring \macro\{(stringA)\}(\{stringB\}) replaces all \{stringA\} to \{stringB\} in \macro.
\sdef \{\{string\}\}(\{parameters\}\}(\{body\}\) behaves like \sdef\{\{string\}\}(\{parameters\}\}(\{body\}\).
\setctable and \restorectable manipulate with stack of catcode tables.
\let \{\{string\}\}(\{stringB\}\) behaves like \let\{\{string\}\}=%\{\{stringB\}\}
\sdef \{\{string\}\}(\{parameters\}\}(\{body\}\) behaves like \sdef\{\{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\}\} \ldots ; repeats \{\{what\}\} \{\{token\}\} for each \{\{token\}\}.

### 1.10 Compatibility with Plain \TeX

All macros of Plain \TeX{} are re-written in Op\TeX{}. Common macros should work in the same sense as in original Plain \TeX{}. Internal control sequences like \f@ot are removed and mostly replaced by control sequences prefixed by _ (like _\this). Only a basic set of old Plain \TeX{} control sequences like \@p, \@s, \@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 \bbox is equal to _bbox. Internal macros of Op\TeX{} have and use only prefixed form. User should use unprefixed forms, but prefixed forms are accessible too because the _ is set as a letter category code globally (in macro files and users document too). Users should re-define unprefixed forms of control sequences without worries that something internal will be broken.

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

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

The text accents macros \^, \^, \^, \^, \^, \^, \^ are undefined\(^\text{8}\) in Op\TeX{}. Use real letters like á, í, ź 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 1 in margins but as A4 with 2.5 cm margins. You can change it, for example by \makepage/1 letter (1,1,1,1)in. This example sets the classical Plain \TeX{} page layout.

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

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

The \sect macro is reserved for sections but original Plain \TeX{} declares this control sequence for math secant\(^\text{9}\).

### 1.11 Related documents

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

\[^{8}\text{The math accents macros like } \acute{a}, \bar{a}, \dot{a}, \hat{a} \text{ still work.}\]
\[^{9}\text{Use } \use\secant(x) \text{ to get } \sec(x).\]
Chapter 2

Technical documentation

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

```latex
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

```latex
optex optex-doc.tex
```

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

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

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

% Catcodes:
\catcode `\{=1 % left brace is begin-group character
\catcode `\}=2 % right brace is end-group character
\catcode `\$=3 % dollar sign is math shift
\catcode `\&=4 % ampersand is alignment tab
\catcode `\#=6 % hash mark is macro parameter character
\catcode `\^^K=7 % circumflex and uparrow are for superscripts
\catcode `\^^A=8 % downarrow is for subscripts
\catcode `\"K=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 127=12 % normal character
```

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 T\TeX. 

The \optexversion and \fmtname are defined.

```ini
\def\optexversion{1.08 Aug 2022}
\def\fmtname{OpTeX}
\let\fmtversion=\optexversion
```

We check if LuaT\TeX\ engine is used at -ini state. And the \~\^J character is set as \newlinechar.
The basic macros for macro file syntax is defined, i.e. \endcode, \doc
and \cod. The \codedecl
will be re-defined later.

Individual *.opm macro files are read.

The file \texttt{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 OpTeX are used and defined with _ prefix. The user can be sure that when he/she does \def\foo then neither internal macros of OpTeX nor \TeX primitives will be damaged. For example \def\if{\ldots} will not damage macros because OpTeX’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.

OpTeX 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 OpTeX macros as internal. User can redefine such prefixed control sequence only if he/she explicitly knows what happens.

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

OpTeX 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 OpTeX, 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 unprefixe 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 OpTeX macros.\footnote{The token \par is in user name space too from OpTeX 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, OpTeX internal macros and packages internal macros.
- Multiletter control sequences in the form \langle\word\rangle_ or \langle\word\rangle_\langle\one\-letter\rangle , where \langle\word\rangle is a sequence of letters, are inaccessible, because they are interpreted as \langle\word\rangle followed by _ or as \langle\word\rangle followed by _\langle\one\-letter\rangle . This is important for writing math, for example:
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 \mathson. 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

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

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

\_codedecl \sequence {Name <version>}

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

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

2.2.4 Name spaces for package writers

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

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

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

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

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

2.2.5 Summary about rules for external macro files published for OpT\TeX

If you are writing a macro file that is intended to be published for OpT\TeX, 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 Op\TeX{} and primitive namespace in read-only mode, if there is not an explicit and documented reason to redefine them.
• Use \namespace{\pkg}⟨name⟩ for your internal macros or \ext{⟨name⟩} if the \namespace{\pkg} is declared. See section 2.2.4.
• Use \load{} (or better: \load*) for loading more external macros if you need them. Don’t use \input{} explicitly in such cases. The reason is: the external macro file is not loaded twice if another macro or the user needs it explicitly too.
• Use \_\texttt{⟨pkg⟩}_{⟨name⟩} for your internal macros or \_.\texttt{⟨name⟩} if the \namespace{\pkg}{} is declared.
• You can write more documentation after the \endcode{} command.
• The Op\TeX{} 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{.opm} differs from file names used directly in Op\TeX{} and from other published macros. This extension \filename{.opm} has precedence before \filename{.tex} when the \load{} macro is used.

The \filename{qrcode.opm} is the first example of how an external macro file for Op\TeX{} can look like.

### 2.2.6 The implementation of the namespaces

```latex
\namespace{\codelen}{\public{\{Prefixing and code syntax <2021-08-16}\}}} % preloaded in format

All \TeX{} primitives have alternative control sequence \hbox \string, ...

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

\namespace{\\\texttt{\public{\(\text{\texttt{sequence}}\text{\texttt{sequence}}\ldots\)} \texttt{\let\texttt{\\texttt{sequence}} = \_\texttt{\texttt{sequence}}} for all sequences.}
\namespace{\\\texttt{\private{\(\text{\texttt{sequence}}\text{\texttt{sequence}}\ldots\)} \texttt{\let\texttt{\private{\texttt{sequence}}} = \texttt{\\texttt{sequence}}} for all sequences.}
\namespace{\\\texttt{\checkexists{\(\text{\texttt{where}}\text{\texttt{prefix}}\text{\texttt{sequence}}} \texttt{\checkexists{\texttt{where}}\texttt{prefix}} prints error if the control sequence propagated to a new name space by \\texttt{public} etc. macros is not declared.}
\namespace{\\\texttt{\xargs{\(\text{\texttt{what}}\text{\texttt{sequence}}\text{\texttt{sequence}}} \texttt{\xargs{\texttt{what}}\texttt{sequence}} for each sequences.}}
```

Each macro file should begin with \_codedecl \macro \{(info)\}. If the \macro is defined already then the \endpinput protects to read such file more than once. Else the \{info\} is printed to the terminal and the file is read.

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

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

The \_mathsbon and \_mathsboff are defined in math-macros.opm file. Now, we define the macros \namespace \{(pkg label)\}, \resetnamespace \{(pkg label)\}, \endnamespace, \nspublic and \nsprivate for package writers, see section 2.2.4.

2.3 pdf\TeX initialization

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

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

\_protected\_def \_pdftexversion {\_numexpr 140 \relax}
\_protected\_def \_pdftexrevision {7}
\_protected\_def \_pdflastlink {\_numexpr \pdffeedback lastlink \relax}
\_protected\_def \_pdfretval {\_numexpr \pdffeedback retval \relax}
\_protected\_def \_pdflastobj {\_numexpr \pdffeedback lastobj \relax}
\_protected\_def \_pdflastannot {\_numexpr \pdffeedback lastannot \relax}
\_def \_pdfxformname {\_pdffeedback xformname}
\_def \_pdfcreationdate {\_pdffeedback creationdate}
\_def \_pdffontname {\_pdffeedback fontname}
\_def \_pdffontobjnum {\_pdffeedback fontobjnum}
\_def \_pdffontsize {\_pdffeedback fontsize}
\_def \_pdfpageref {\_pdffeedback pageref}
\_def \_pdfcolorstackinit {\_pdffeedback colorstackinit}
\_protected\_def \_pdfliteral {\_pdftexextension \pdfliteral}
\_protected\_def \_pdfcolorstack {\_pdftexextension \pdfcolorstack}
\_protected\_def \_pdfnetmatrix {\_pdftexextension \pdfnetmatrix}
\_protected\_def \_pdfsave {\_pdftexextension \pdfsave \relax}
\_protected\_def \_pdfrestore {\_pdftexextension \pdfrestore \relax}
\_protected\_def \_pdfobj {\_pdftexextension \pdfobj \relax}
\_protected\_def \_pdfrefobj {\_pdftexextension \pdfrefobj \relax}
\_protected\_def \_pdffont{
}
\_protected\_def \_pdfextlink{
}
\_protected\_def \_pdfstartthread{
}
\_protected\_def \_pdfstartthread{
}
We define first bundle of basic macros.

\begin{verbatim}
\_protected\_def \_pdfendthread \_pdfextension endthread\_relax
\_protected\_def \_pdfinfo \_pdfextension info
\_protected\_def \_pdfcatalog \_pdfextension catalog
\_protected\_def \_pdfnames \_pdfextension names
\_protected\_def \_pdfincludelchars \_pdfextension includechars
\_protected\_def \_pdffontattr \_pdfextension fontattr
\_protected\_def \_pdfmapfile \_pdfextension mapfile
\_protected\_def \_pdfmapline \_pdfextension mapline
\_protected\_def \_pdftrailer \_pdfextension trailer
\_protected\_def \_pdfglyphtounicode \_pdfextension glyphtounicode
\_protected\_edef \_pdfcompresslevel \_pdfvariable compresslevel
\_protected\_edef \_pdfobjcompresslevel \_pdfvariable objcompresslevel
\_protected\_edef \_pdfdecimaldigits \_pdfvariable decimaldigits
\_protected\_edef \_pdfgamma \_pdfvariable gamma
\_protected\_edef \_pdfimageresolution \_pdfvariable imageresolution
\_protected\_edef \_pdfimageapplygamma \_pdfvariable imageapplygamma
\_protected\_edef \_pdfimagemegamma \_pdfvariable imagemegamma
\_protected\_edef \_pdfimagehicolor \_pdfvariable imagehicolor
\_protected\_edef \_pdfimageaddfilename \_pdfvariable imageaddfilename
\_protected\_edef \_pdfpkresolution \_pdfvariable pkresolution
\_protected\_edef \_pdfincludefilename \_pdfvariable includefilename
\protected\_edef \_pdfminorversion \_pdfvariable minorversion
\protected\_edef \_pdfuniqueresname \_pdfvariable uniqueresname
\_pdfhorigin \_pdfvariable horigin
\_pdfvorigin \_pdfvariable vorigin
\_pdflinkmargin \_pdfvariable linkmargin
\_pdfdestmargin \_pdfvariable destmargin
\_pdfthreadmargin \_pdfvariable threadmargin
\_pdfpagesattr \_pdfvariable pagesattr
\_pdfpageattr \_pdfvariable pageattr
\_pdfpageresources \_pdfvariable pageresources
\_pdfxformattr \_pdfvariable xformattr
\_pdfxformresources \_pdfvariable xformresources
\_pdfpkmode \_pdfvariable pkmode
\end{verbatim}

2.4 Basic macros

We define first bundle of basic macros.

\begin{verbatim}
\_protected\_def \_pdfbegin \_pdfextension begin\_relax
\_protected\_def \_pdfend \_pdfextension end\_relax
\_protected\_def \_pdfbeginthread \_pdfextension beginthread\_relax
\_protected\_def \_pdfendthread \_pdfextension endthread\_relax
\_protected\_def \_pdfinfo \_pdfextension info
\_protected\_def \_pdfcatalog \_pdfextension catalog
\_protected\_def \_pdfnames \_pdfextension names
\_protected\_def \_pdfincludelchars \_pdfextension includechars
\_protected\_def \_pdffontattr \_pdfextension fontattr
\_protected\_def \_pdfmapfile \_pdfextension mapfile
\_protected\_def \_pdfmapline \_pdfextension mapline
\_protected\_def \_pdftrailer \_pdfextension trailer
\_protected\_def \_pdfglyphtounicode \_pdfextension glyphtounicode
\_protected\_edef \_pdfcompresslevel \_pdfvariable compresslevel
\_protected\_edef \_pdfobjcompresslevel \_pdfvariable objcompresslevel
\_protected\_edef \_pdfdecimaldigits \_pdfvariable decimaldigits
\_protected\_edef \_pdfgamma \_pdfvariable gamma
\_protected\_edef \_pdfimageresolution \_pdfvariable imageresolution
\_protected\_edef \_pdfimageapplygamma \_pdfvariable imageapplygamma
\_protected\_edef \_pdfimagemegamma \_pdfvariable imagemegamma
\_protected\_edef \_pdfimagehicolor \_pdfvariable imagehicolor
\_protected\_edef \_pdfimageaddfilename \_pdfvariable imageaddfilename
\_protected\_edef \_pdfpkresolution \_pdfvariable pkresolution
\_protected\_edef \_pdfincludefilename \_pdfvariable includefilename
\protected\_edef \_pdfminorversion \_pdfvariable minorversion
\protected\_edef \_pdfuniqueresname \_pdfvariable uniqueresname
\_pdfhorigin \_pdfvariable horigin
\_pdfvorigin \_pdfvariable vorigin
\_pdflinkmargin \_pdfvariable linkmargin
\_pdfdestmargin \_pdfvariable destmargin
\_pdfthreadmargin \_pdfvariable threadmargin
\_pdfpagesattr \_pdfvariable pagesattr
\_pdfpageattr \_pdfvariable pageattr
\_pdfpageresources \_pdfvariable pageresources
\_pdfxformattr \_pdfvariable xformattr
\_pdfxformresources \_pdfvariable xformresources
\_pdfpkmode \_pdfvariable pkmode
\end{verbatim}

\begin{verbatim}
\_protected\_edef \_pdfcompresslevel \_pdfvariable compresslevel
\_protected\_edef \_pdfobjcompresslevel \_pdfvariable objcompresslevel
\_protected\_edef \_pdfdecimaldigits \_pdfvariable decimaldigits
\_protected\_edef \_pdfgamma \_pdfvariable gamma
\_protected\_edef \_pdfimageresolution \_pdfvariable imageresolution
\_protected\_edef \_pdfimageapplygamma \_pdfvariable imageapplygamma
\_protected\_edef \_pdfimagemegamma \_pdfvariable imagemegamma
\_protected\_edef \_pdfimagehicolor \_pdfvariable imagehicolor
\_protected\_edef \_pdfimageaddfilename \_pdfvariable imageaddfilename
\_protected\_edef \_pdfpkresolution \_pdfvariable pkresolution
\_protected\_edef \_pdfincludefilename \_pdfvariable includefilename
\protected\_edef \_pdfminorversion \_pdfvariable minorversion
\protected\_edef \_pdfuniqueresname \_pdfvariable uniqueresname
\_pdfhorigin \_pdfvariable horigin
\_pdfvorigin \_pdfvariable vorigin
\_pdflinkmargin \_pdfvariable linkmargin
\_pdfdestmargin \_pdfvariable destmargin
\_pdfthreadmargin \_pdfvariable threadmargin
\_pdfpagesattr \_pdfvariable pagesattr
\_pdfpageattr \_pdfvariable pageattr
\_pdfpageresources \_pdfvariable pageresources
\_pdfxformattr \_pdfvariable xformattr
\_pdfxformresources \_pdfvariable xformresources
\_pdfpkmode \_pdfvariable pkmode
\end{verbatim}

3 \_protected\_def \_pdfminorversion \_pdfvariable minorversion
3 \_protected\_def \_pdfobjcompresslevel \_pdfvariable objcompresslevel
3 \_protected\_def \_pdfcompresslevel \_pdfvariable compresslevel
3 \_protected\_def \_pdfpkresolution \_pdfvariable pkresolution
3 \_protected\_def \_pdfincludefilename \_pdfvariable includefilename
3 \protected\_edef \_pdfminorversion \_pdfvariable minorversion
3 \protected\_edef \_pdfuniqueresname \_pdfvariable uniqueresname
3 \_pdfhorigin \_pdfvariable horigin
3 \_pdfvorigin \_pdfvariable vorigin
3 \_pdflinkmargin \_pdfvariable linkmargin
3 \_pdfdestmargin \_pdfvariable destmargin
3 \_pdfthreadmargin \_pdfvariable threadmargin
3 \_pdfpagesattr \_pdfvariable pagesattr
3 \_pdfpageattr \_pdfvariable pageattr
3 \_pdfpageresources \_pdfvariable pageresources
3 \_pdfxformattr \_pdfvariable xformattr
3 \_pdfxformresources \_pdfvariable xformresources
3 \_pdfpkmode \_pdfvariable pkmode

\begin{verbatim}
\_protected\_edef \_pdfcompresslevel \_pdfvariable compresslevel
\_protected\_edef \_pdfobjcompresslevel \_pdfvariable objcompresslevel
\_protected\_edef \_pdfdecimaldigits \_pdfvariable decimaldigits
\_protected\_edef \_pdfgamma \_pdfvariable gamma
\_protected\_edef \_pdfimageresolution \_pdfvariable imageresolution
\_protected\_edef \_pdfimageapplygamma \_pdfvariable imageapplygamma
\_protected\_edef \_pdfimagemegamma \_pdfvariable imagemegamma
\_protected\_edef \_pdfimagehicolor \_pdfvariable imagehicolor
\_protected\_edef \_pdfimageaddfilename \_pdfvariable imageaddfilename
\_protected\_edef \_pdfpkresolution \_pdfvariable pkresolution
\_protected\_edef \_pdfincludefilename \_pdfvariable includefilename
\protected\_edef \_pdfminorversion \_pdfvariable minorversion
\protected\_edef \_pdfuniqueresname \_pdfvariable uniqueresname
\_pdfhorigin \_pdfvariable horigin
\_pdfvorigin \_pdfvariable vorigin
\_pdflinkmargin \_pdfvariable linkmargin
\_pdfdestmargin \_pdfvariable destmargin
\_pdfthreadmargin \_pdfvariable threadmargin
\_pdfpagesattr \_pdfvariable pagesattr
\_pdfpageattr \_pdfvariable pageattr
\_pdfpageresources \_pdfvariable pageresources
\_pdfxformattr \_pdfvariable xformattr
\_pdfxformresources \_pdfvariable xformresources
\_pdfpkmode \_pdfvariable pkmode
\end{verbatim}

2.4 Basic macros

We define first bundle of basic macros.
\_let \_bgroup= \_let \_egroup= \_def \_empty {} \_def \_space { } \_def \_null { \_hbox{} } \_public \bgroup \egroup \empty \space \null ;

\ignoreit ignores next token or \{\text\}, \useit{\text} expands to \text (removes outer braces), \ignoresecond uses first, ignores second parameter and \usesecond ignores first, uses second parameter.

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

\bslash is “normal backslash” with category code 12. \nbb is double backslash and \pcent is normal %. They can be used in Lua codes, for example.

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

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

\sxdef {\langle text\rangle} is equivalent to \xdef\langle text\rangle.

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

\adef {\langle char\rangle}{\langle body\rangle} puts the \langle char\rangle as active character and defines it as \{\langle body\rangle\}. You can declare a macro with parameters too. For example \adef @#1{...#1...}.

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

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

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

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

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

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

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

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

2.5 Allocators for TeX registers

Like plainTeX, the allocators \newcount, \newwrite, etc. are defined. The registers are allocated from 256 to the \_mai\langle type \rangle which is 65535 in LuaTeX.

Unlike in PlainTeX, the mentioned allocators are not \outer.

User can use \dimen0 to \dimen200 and similarly for \skip, \muskip, \box, and \toks directly.

User can use \count20 to \count200 directly too. This is the same philosophy as in old plainTeX, but the range of directly used registers is wider.

Inserts are allocated from 254 to 201 using \newinsert.

You can define your own allocation concept (for example for allocation of arrays) from the top of the registers array. The example shows a definition of the array-like declarator of counters.

\newcount \_maicount % redefine maximal allocation index as variable
\_maicount = \maicount % first value is top of the array
\def\newcountarray #1[#2]{% \newcountarray \foo[100]
  \global\advance\_maicount by -#2\relax
  \ifnum\_countalloc > \_maicount
    \errmessage{No room for a new array of \string\count}%
  \else
    \global\chardef#1=#\_maicount
  \fi
}
\def\usecount #1[#2]{% \usecount \foo[2]
  \count\numexpr#1+#2\relax
}

The limits are set first.

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

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

The \texttt{\newinsert} macro is defined differently than others.

Other allocation macros \texttt{\newmarks}, \texttt{\newattribute} and \texttt{\newcatcodetable} have their counter allocated by the \texttt{\newmacro} macro. \texttt{\noattr} is constant \texttt{\textasciitilde7FFFFFFF}, i.e. unused attribute
We declare public and private versions of \tmpnum and \tmpdim registers separately. They are independent registers.

A few registers are initialized like in plain TeX. We absolutely don’t support the @category dance, so \z@skip \z@, \p@ etc. are not defined in OpTEX. If you need such control sequences then you can initialize them by \load[plain-at]. Only the \zo and \zoskip (equivalents to \z@ and \z@skip) are declared here and used in some internal macros of OpTEX for improving speed.

### 2.6 If-macros, loops, is-macros

#### 2.6.1 Classical \newif

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

The macro \newif enables to declare \ifxxx and to use \xxxtrue and \xxxfalse. This means that it is usable for the internal namespace (_prefixed macros).

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

\if<something> \afterfi{⟨result is true⟩} \else \afterfi{⟨result is false⟩} \fi
2.6.2 Loops

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

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

The `\_loop` macro keeps its original plain TeX meaning. It is not expandable and nested `\_loops` are possible only in a TeX group.

```latex
\_long\_def \_loop #1\_repeat{\_def \_body(#1)\_iterate}
\_let \_repeat=\_fi % this makes \loop...\if...\repeat skippable
\_let \_repeat=\_fi
\_def \_iterate {\_body \_ea \_iterate \_fi}
```

```
\_foreach \{list\} \do \{what\} repeats \{what\} for each element of the \{list\}. The \{what\} can include \#1 which is substituted by current number. 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:
```

```latex
\_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.\" \}
```

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.

```latex
\_newcount \frnum % the numeric variable used in \_foreach
\_def \_do{\_doundefined} % we need to ask _ifs#1\_do ...
```

```
\_long\_def \_foreach \#1\_do #2\{\_isempty{#2}\_iftrue\_afterfi{\_foreachA{\#1}{##1}}\_else\_afterfi{\_foreachA{\#1}{#2}}\_fi}
```

```
\_foreach \{from\}...\{to\} \do \{what\} or \_foreachstep \{num\} \{from\}...\{to\} \do \{what\} repeats \{what\} for each number from \{from\} to \{to\} (with step \{num\}) or with step one. The \{from\} 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 \_foreachB 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 cryptic TeX language.

```latex
\_def \_foreachB \#1\#2\do{\_foreachstep 1:\#1..\#2\_do}
```

```
\_long\_def \_foreachstep \#1..\#2..\#3\_do#4{\_putforstack
\_immediateassignment\%\_global\_frnum=\_numexpr\#2-\_relax\%
}\_ea\_foreachB\_ea{\_the\_numexpr\#3\_ea}\_ea{\_the\_numexpr\#1\_ea}
```

```latex
\_def \_do{\_doundefined}
```

### 2.6.2 Loops

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

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

The `\_loop` macro keeps its original plain TeX meaning. It is not expandable and nested `\_loops` are possible only in a TeX group.

```latex
\_long\_def \_loop #1\_repeat{\_def \_body(#1)\_iterate}
\_let \_repeat=\_fi % this makes \loop...\if...\repeat skippable
\_let \_repeat=\_fi
\_def \_iterate {\_body \_ea \_iterate \_fi}
```

```
\_foreach \{list\} \do \{what\} repeats \{what\} for each element of the \{list\}. The \{what\} can include \#1 which is substituted by current number. 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:
```

```latex
\_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.\" \}
```

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.

```latex
\_newcount \frnum % the numeric variable used in \_foreach
\_def \_do{\_doundefined} % we need to ask _ifs#1\_do ...
```

```
\_long\_def \_foreach \#1\_do #2\{\_isempty{#2}\_iftrue\_afterfi{\_foreachA{\#1}{##1}}\_else\_afterfi{\_foreachA{\#1}{#2}}\_fi}
```

```
\_foreach \{from\}...\{to\} \do \{what\} or \_foreachstep \{num\} \{from\}...\{to\} \do \{what\} repeats \{what\} for each number from \{from\} to \{to\} (with step \{num\}) or with step one. The \{from\} 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 \_foreachB 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 cryptic TeX language.

```latex
\_def \_foreachB \#1\#2\do{\_foreachstep 1:\#1..\#2\_do}
```

```
\_long\_def \_foreachstep \#1..\#2..\#3\_do#4{\_putforstack
\_immediateassignment\%\_global\_frnum=\_numexpr\#2-\_relax\%
}\_ea\_foreachB\_ea{\_the\_numexpr\#3\_ea}\_ea{\_the\_numexpr\#1\_ea}
```

```latex
\_def \_do{\_doundefined}
```

```latex
\_newcount \frnum % the numeric variable used in \_foreach
\_def \_do{\_doundefined} % we need to ask _ifs#1\_do ...
```
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.

```
\_newcount\_forlevel
\_def\_putforstack{\_immediateassigned{\_ifnum\_forlevel>0
\_sxdef{\_frnum:\_the\_forlevel\_ea}{\_the\_frnum}\%
\_global\_slet{\_fbody:\_the\_forlevel}{\_fbody}\%
\_fi
\_incr\_forlevel
}\_ifnum\_forlevel>0
\_global\_slet{\_fbody}{\_fbody:\_the\_forlevel}\%
\_global\_frnum=\_cs{\_frnum:\_the\_forlevel}\_space
\_fi
}
\_def\_getforstack{\_immediateassigned{\_decr\_forlevel
\_ifnum\_forlevel>0\_global\_slet{\_fbody}{\_fbody:\_the\_forlevel}\%
\_global\_frnum=\_cs{\_frnum:\_the\_forlevel}\_space
\_fi
}}
\_ifx\_immediateassignment\_undefined % for compatibility with older \LuaTeX
\_let\_immediateassigned=\_useit \_let\_immediateassignment=\_empty
\_fi
```

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.

```
\_def\_foreachdef\#1\#2\#{\_toks0{\#2}\
\_long\_edef\#1##1{\_ea\_noexpand\csname _body:\_csstring#1\_endcsname
##1\_the\_toks0 \_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 ;
```

2.6.3 Is-macros

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

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

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

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

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

```
\isempty \{\langle 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.
```

43
\_isempty \_if\_relax\_detokenize\(\#1\)\_relax \_else \_ea\_unless \_fi\#2
\_istoksempty \_ea\_isempty\_ea\(\_the\#1\)\#2
\isempty \istoksempty ;
\isequal {\langle \text{textA} \rangle}{\langle \text{textB} \rangle}\text{if true if the }\langle \text{textA} \rangle\text{ and }\langle \text{textB} \rangle\text{ are equal, only from strings point of view, category codes are ignored. The macro is expandable.}
\_isequal\#1\#2\#3\{\_directlua{\_luaescapestring{\_detokenize{\#1}}}==\_luaescapestring{\_detokenize{\#2}}\text{ then else tex.print("\nbb unless") end}#3\}
\isempty \ismacro \macro{\text} \text{is true if macro is defined as }\langle \text \rangle. \text{ Category codes are ignored in this testing. The macro is expandable.}
\isdefined \{\csname \text \}\text{is true if }\csname \text \csname \endcsname \_ea\_unless \_fi \#2\}
\ismacro \ismacro\{\text\} \text{is true if macro is defined as }\langle \text \rangle. \text{ The macro is expandable.}
\isinlist \list\{\text\} \text{is true if the }\langle \text \rangle\text{ is included the macro body of the }\list. \text{ The category codes are relevant here. The macro is expandable.}
\isfile \{\filename\} \text{is true if the file }\langle \filename \rangle\text{ exists and are readable by TeX.}
\isfont \{\fontname or [fontfile]\} \text{is true if a given font exists. The result of this testing is saved to the }\_ifexistfam. \text{ The last macro }\isnextchar \text{has a different syntax than all other is-macros. It executes }\langle \text{codeA} \rangle\text{ if next character is equal to }\langle \text{char} \rangle. \text{ Else the }\langle \text{codeB} \rangle\text{ is executed. The macro is not expandable.}
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 OpTeX macros.

Both groups of registers have their type: number, dimension, skip, token list.

The registers are represented by their names (control sequences). If the user re-defines this control sequence then the appropriate register exists steadily and build-in algorithms are using it without change. But user cannot access its value in this case. OpTeX declares two control sequences for each register: prefixed (private) and unprefixed (public). OpTeX 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 OpTeX macros still work without change.

2.7.1 Primitive registers

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

```latex
\parindent=20pt % indentation of paragraphs
\pretolerance=100 % parameters used in paragraph breaking algorithm
\tolerance=200
\badness=1000
\doublehyphenpenalty=10000
\finalhyphenpenalty=5000
\adjdemerits=10000
\uchyph=1
\defaulthyphenchar=`\-
\defaultskewchar=-1
hfuzz=0.1pt
\vfuzz=0.1pt
\overfullrule=5pt % penalty between lines inside the paragraph
\linenumberpenalty=10 % when a word is broken
\hyphenpenalty=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
\nulldelimiterspace=1.2pt
\scriptspace=0.5pt % \mathspaceafterscript used in \setmathdimens, \setunimathdimens instead
\splitmaxdepth=4pt
\splitmaxdepth=\maxdimen
\boxmaxdepth=\maxdimen
\parskip=0pt plus 1pt
\abovedisplayskip=12pt plus 3pt minus 9pt
\abovedisplayskipskip=0pt plus 3pt
\belowdisplayskip=12pt plus 3pt minus 9pt
\belowdisplayskipskip=7pt plus 3pt minus 4pt
\parfillskip=0pt plus 1fil
\thickmuskip=3mu
\medmuskip=4mu plus 2mu minus 4mu
\thickmuskip=5mu plus 5mu
\topskip=10pt % top edge of page-box to first baseline distance
\splittopskip=10pt % top edge of page-box to first baseline distance
```

Note that \topskip and \splittopskip are changed when first \typsize sets the main values (default font size and default \baselineskip).

45
2.7.2 Plain TeX registers

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

% We also define special registers that function like parameters:
\_newskip\_smallskipamount \_smallskipamount=3pt plus 1pt minus 1pt
\_newskip\_medskipamount \_medskipamount=6pt plus 2pt minus 2pt
\_newskip\_bigskipamount \_bigskipamount=12pt plus 4pt minus 4pt
\_newskip\_normalbaselineskip \_normalbaselineskip=12pt
\_newskip\_normallineskip \_normallineskip=1pt
\_newdimen\_normallineskiplimit \_normallineskiplimit=0pt
\_newdimen\_jot \_jot=3pt
\_newcount\_interdisplaylinepenalty \_interdisplaylinepenalty=100
\_newcount\_interfootnotelinepenalty \_interfootnotelinepenalty=100
\_public \smallskipamount \medskipamount \bigskipamount
\normalbaselineskip \normallineskip \normallineskiplimit
\jot \interdisplaylinepenalty \interfootnotelinepenalty;

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

% We want to use third pass of paragraph building algorithm
% we don’t need compatibility with old documents
\_emergencystretch=20pt
\_clubpenalty=10000 % after first line of paragraph
\_widowpenalty=10000 % before last line of paragraph
\_showboxbreadth=150 % for tracing boxes
\_showboxdepth=7
\_errorcontextlines=15
\_tracinglostchars=2 % missing character warnings on terminal too
\_outputmode=1 % PDF output
\_pdfvorigin=0pt % origin is exactly at upper left corner
\_pdfhorigin=0pt
\_hoffset=25mm % margins are 2.5cm, no 1in
\_voffset=25mm
\_hsize=160mm % 210mm (from A4 size) – 2*25mm (default margins)
\_vsize=244mm % 297mm (from A4 size) – 2*25mm (default margins) –3mm baseline correction
\_pagewidth=210 true mm
\_pageheight=297 true mm

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

The main principle of how to configure OpTFX is not to use only parameters. A designer can copy macros from OpTFX and re-define them as required. This is a reason why we don’t implement dozens of parameters, but we keep OpTFX 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 OpTFX (OPmac uses macros for them). The reason of this difference: if a user sets parameter by unprefixed (public) control sequence, an OpTFX 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
tabluter behaves like three spaces in multiline verbatim.

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

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

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

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

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

The \everycapitonf 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 \thisstable 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 \thisstable 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.

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

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

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

\tsize is virtual unit of the width of paragraph-like table items when \table pxto{$\langle$ size $\rangle$} is used.
The \texttt{\textbackslash eqalign} macro can be configured by \texttt{\textbackslash eqlines} and \texttt{\textbackslash eqstyle} tokens lists. The default values are set in order these 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.

\begin{verbatim}
\newtoks \eqlines \eqlines=(\openup\jot)
\newtoks \eqstyle \eqstyle=(\strut\displaystyle)
\newdimen \eqspace \eqspace=20pt
\public \eqlines \eqstyle \eqspace ;
\end{verbatim}

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

\begin{verbatim}
\newtoks \lmfil \lmfil=(\hfil)
\public \lmfil ;
\end{verbatim}

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

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

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

\begin{verbatim}
\newdimen \headlinedist \headlinedist=14pt
\newdimen \footlinedist \footlinedist=24pt
\public \headlinedist \footlinedist ;
\end{verbatim}

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

\begin{verbatim}
\newskip \pgbottomskip \pgbottomskip=0pt \relax
\public \pgbottomskip ;
\end{verbatim}

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:

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

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

\begin{verbatim}
\newtoks \nextpages \nextpages=
\public \nextpages ;
\end{verbatim}

The \texttt{\textbackslash 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:

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

The parameters used in \texttt{\textbackslash inoval} and \texttt{\textbackslash incircle} macros can be re-set by \texttt{\ovalparams, \circleparams} tokens lists. The default values (documented in the user manual) are set in the macros.
OpTeX defines “Standard OpTeX markup language” which lists selected commands from chapter 1 and gives their behavior when a converter from OpTeX document to HTML or Markdown or \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{\textbackslash cnvinfo{\{data\}}} . OpTeX simply ignores this but the converter can read its configuration from here. For example, a user can write:

\begin{verbatim}
\def \cnvinfo {\texttt{\textbackslash \let \cnvinfo = \_ignoreit}}
\end{verbatim}

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

2.8 More OpTeX macros

The second bundle of OpTeX macros is here.

We define \texttt{\textbackslash opinput{\{file name\}}} macro which does \texttt{\textbackslash input{\{file name\}}} but the catcodes are set to normal catcodes (like OpTeX initializes them) and the catcodes setting is returned back to the current values when the file is read. You can use \texttt{\textbackslash 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{\textbackslash optexcatcodes} catcode table and \texttt{\textbackslash plaintexcatcodes} . They save the commonly used catcode tables. Note that \texttt{\textbackslash catcodetable} is a part of Lua\TeX{} extension. The catcodetable stack is implemented by OpTeX macros. The \texttt{\textbackslash setctable{\{catcode table\}}} pushes current catcode table to the stack and activates catcodes from the \{catcode table\} . The \texttt{\textbackslash restorectable} returns to the saved catcodes from the catcode table stack.

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

The implementation of the catcodetable stack follows.

The current catcodes are managed in the \texttt{\_ctablelist} . If the \texttt{\_setctable} is used first (or at the outer level of the stack), then the \texttt{\_ctablelist} 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 \texttt{\_ctablelist} macro. The \texttt{\_restorable} reads the last saved catcode table number from the \texttt{\_ctablelist} and uses it.

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

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

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

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

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

The declarator \eoldef \macro \#1\lbrace \langle replacement text \rangle \rbrace 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 \langle replacement text \rangle. 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 \langle parameter \rangle 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 verbatim scanner of \texttt{\textbackslash scarto\textbackslash eol} scans the \texttt{\textbackslash text to end of line} in verbatim mode and runs the \texttt{\textbackslash macro\{\textbackslash text to end of line\}}. The \texttt{\textbackslash macro} can be defined \texttt{\def \textbackslash macro\{\textbackslash \ldots \textbackslash scantextokens\{\textbackslash \ldots \textbackslash \}}. 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 \texttt{\chap}, \texttt{\sec}, \texttt{\secc} and \texttt{\_Xtoc} macros. It means that user can write \texttt{\_sec text `&` text} for example. Inline verbatim works in title sections.

The verbatim scanner of \texttt{\textbackslash scantro\textbackslash eol} keeps category 7 for ^ in order to be able to use ^ as comment character which means that the next line continues.

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

How it works: \texttt{\textbackslash replstring\textbackslash foo\{\textbackslash textA\}\{\textbackslash textB\}} prepares \texttt{\_replacestrings\#1\{\textbackslash textA\}\ldots\ldots} and runs \texttt{\_replacestrings\#1\{\textbackslash foo-body\}\{\textbackslash textA\}\{\textbackslash textA\}}. So, \#1 includes the first part of \texttt{\textbackslash foo-body} before first \texttt{\textbackslash textA}. It is saved to \texttt{\_tmptoks} and \texttt{\_replacestrings\#1\{\textbackslash textA\}\textbackslash \_replacestrings\#1\{\textbackslash textB\}} is run in a loop. It finishes processing or appends the next part to \texttt{\_tmptoks} separated by \texttt{\textbackslash textB} and continues loop. The final part of the macro removes the last \texttt{\textbackslash \_scanto\textbackslash eol} from resulting \texttt{\_tmptoks} and defines a new version of the \texttt{\textbackslash foo}. The \texttt{\_catcode} primitive is redefined here. Why? There is very common cases like \texttt{\_catcode\textbackslash ^\{\_something\}} or \texttt{\_catcode\textbackslash ^\{\_number\}} but these characters ^ or ^ can be set as active (typically by \texttt{\verbchar} macro). Nothing problematic happens if re-defined \texttt{\_catcode} is used in this case.

If you really need primitive \texttt{\_catcode} then you can use \texttt{\_catcode}. The \texttt{\_removespaces \{\texttt{\textbackslash text with spaces}\}} \} expands to \texttt{\textbackslash textwithoutspaces}. The \texttt{\_ea\_ignorept\texttt{\textbackslash the\{dimen\}}} expands to a decimal number \texttt{\textbackslash the\{dimen\}} but without \texttt{pt} unit.
You can use expandable \bp{dimen} convertor from \TeX{ dimen} (or from an expression accepted by \dimexpr primitive) to a decimal value in big points (used as natural unit in the PDF format). So, you can write, for example:

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

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

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

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

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

```
\_doc \langle ignored text \rangle \langle documentation \rangle \_cod \langle ignored text \rangle
```

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

```
\_long\_def\_doc #1\_cod {{skiptoeol}}
```

### 2.9 Using key=value format in parameters

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

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

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

or

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

or simply \myframe {text3}. You can define \myframe as follows:

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

We recommend using \optdef for defining macros with optional parameters written in \[]. Then the optional parameters are saved in the \opt tokens register. First: we 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{\langle key \rangle} macro. The \kv{\langle key \rangle} 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 [..., draft, ...]{text} 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{\langle list \rangle} 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{\langle key \rangle} macros. The \kv{\langle key \rangle} expands the \kv{\langle key \rangle} 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. The \magstep and \magstephalf are defined with \space, (no \relax), in order to be expandable.

Plain \TeX basic macros and control sequences. \endgraf, \endline. The ^^L is not defined in Op\TeX because it is obsolete.
\def"M{} % control <return> = control <space>
\def"I{} % same for <tab>
\_def\^{\ }% control <return> = control <space>
\_def\^{\ }% same for <tab>
\_def\lq{`}\def\rq{'}
\_def\lbrack{[}\_def\rbrack{]}
% They are only public versions.
% \catcode\^^L=active \outer\def\^^L{\par} % ascii form-feed is "\outer\par" obsolete
\_let\_endgraf=\_par \_let\_endline=\_cr
\_public\_endgraf\_endline;

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

\_protected\_def\_thinspace{\_kern.16667em}
\_protected\_def\_negthinspace{\_kern-.16667em}
\_protected\_def\_enspace{\_kern.5em}
\_protected\_def\_enskip{\_hskip.5em\_relax}
\_protected\_def\_quad{\_hskip1em\_relax}
\_protected\_def\_qquad{\_hskip2em\_relax}
\_protected\_def\_smallskip{\_vskip\_smallskipamount}
\_protected\_def\_medskip{\_vskip\_medskipamount}
\_protected\_def\_bigskip{\_vskip\_bigskipamount}
\_def\_nointerlineskip{\_prevdepth=-1000pt}
\_def\_offinterlineskip{\_baselineskip=-1000pt\_lineskip=0pt\_lineskiplimit=\_maxdimen}
\_public\_thinspace\_negthinspace\_enspace\_enskip\_quad\_smallskip\_medskip\_bigskip\_nointerlineskip\_offinterlineskip;

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

Spaces. \verb|\thinspace|, \verb|\negthinspace|, \verb|\enspace|, \verb|\enskip|, \verb|\quad|, \verb|\medskip|, \verb|\bigskip|, \verb|\nointerlineskip|, \verb|\offinterlineskip|, \verb|\topglue|, \verb|\vglue|, \verb|\hglue|, \verb|\slash|.
Boxes. \line, \leftline, \rightline, \centerline, \rlap, \llap, \underbar.

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

Alignment. \hidewidth \ialign \multispan.

Tabbing macros are omitted because they are obsolete.

Indentation and others. \textindent, \item, \itemitem, \narrower, \raggedright, \tttraggedright, \leavevmode.

Few character codes are set for backward compatibility. But old obscurities (from plain TeX) based on \mathhexbox are not supported – an error message and recommendation to directly using the desired character is implemented by the \usedirectly macro). The user can re-define these control sequences of course.
The \unichars macro is run in \initunifonts, Unicodes are used instead old plain \TeX settings.

Accents. The macros \oalign, \d, \b, \c, \dots, are defined for backward compatibility.

The accent commands like \v, \., \H, etc. are not defined. Use the accented characters directly – it is the best solution. But you can use the macro \oldaccents which defines accented macros. Much more usable is to define these control sequences for other purposes.
The plain TeX macros \hrulefill, \dotfill, \rightarrowfill, \leftarrowfill, \downbracefill, \upbracefill. The last four are used in non-Unicode variants of \overrightarrow, \overleftarrow, \overbrace and \underbrace macros, see section 2.15.

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

Plain TeX reads hyphen.tex with patterns as \language=0. We do the same.
2.11 Preloaded fonts for text mode

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

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

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

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 OpTeX declares Unicode fonts. You can create such fonts-less format anyway if you set \fontspreload to \relax before \input optex.ini, i.e.: luatex --ini '-let\fontspreload=\relax \input optex.ini'

2.12 Using \font primitive directly

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

\font \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.tfm (and additional data for glyphs) must be correctly installed in your system. If you want to load OpenType otf or ttf font files, use the declarator \initunifonts. This command adds additional features to the \font primitive which gives the extended syntax:
\font \{\font switch\} = \{(font file name)\} : \{font features\} \ (size spec)
% or
\font \{\font switch\} = \{(font name)\} : \{font features\} \ (size spec)

where \(\text{font file name}\) 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 \(\text{font file name}\) or \(\text{font name}\) includes spaces. The original syntax for tfm files is also available. Example:

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

\initunifonts loads the implementation of the \font primitive from luaotfload package. More information is available in the \texttt{luaotfload-latex.pdf} file.

Note, that \fontfam does (among other things) \initunifonts internally. You need not to specify it if \fontfam 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 \{\size spec\} saves the information about \size spec. This information is taken into account when a variant selector (for example \texttt{\rm, \bf, \it, \bi}) or \resizethefont is used. The \size spec can be:

- \texttt{at\langle dimen\rangle}, for example \setfontsize{at12pt}. It gives the desired font size directly.
- \texttt{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\... scaled\langle number\rangle.
- \texttt{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:

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

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

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

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

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

\initunifonts
\font\tencrimson=[Crimson-Roman]:+onum \% font Crimson-Regular at 10 pt is loaded
\def\crimson{\tencrimson\resizethefont} \% crimson uses the font size on demand

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

This is not only an academical example. The \texttt{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.
2.12.2 The %fontlet declarator

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

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

example:
\fontlet \bigfont = \_tenbf at15pt

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

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

2.12.3 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\langle dimen \rangle or mag\langle factor \rangle. Then the command \_resizethefont or variant selectors try to choose the font suitable for the required optical size. For example

\fontfam[lm]
\_setfontsize{at13pt}\rm
\_setfontsize\(\text{at}13\text{pt}\)\rm
Now in the text is printed in \langle lmroman12-regular \rangle at 13 pt.

See also section 2.13.12.

2.12.4 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 luahtex 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=\langle NotoSansDevanagari-Regular \rangle:mode=harf;script=dev2

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

2.12.5 Implementation of resizing

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

\_def\_initunifonts {%
\_directlua{%
require('luaotfload-main')
luaotfload.main()
optex_hook_into_luaotfload()
}%
\_glet \_fmodtt=\_unifmodtt % use \_ttunifont for \tt
\_glet \_initunifonts=\_relax % we need not to do this work twice
\_glet \_initunifonts=\_relax
}%
\_public \_initunifonts ;
The \setfontsize \{size spec\} macro is re-calculated if an \optsize value is calculated from \sizespec and \setfontsize \defaultoptsize. If the \optsize is in the mag(number) format then \setfontsize \defaultoptsize is calculated from the at\{dimen\} format using previous \optsize value.

\verbatiminput{fonts-resize.opm}

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

\verbatiminput{fonts-resize.opm}

The \fontlet \{font switch A\} \{font switch B\} \size spec does

\verbatiminput{fonts-resize.opm}

\newcurrfontsize \size spec does \fontlet \{saved switch\} \font \size spec \relax \{saved switch\}. It changes the current font at the given \size spec.

\verbatiminput{fonts-resize.opm}

The \regtfm \fontid \{optical size data\} registers optical sizes data directly by the font file names. This can be used for tfm files or OpenType files without various font features. See also \regoptsizes in section 2.13.12. The \regtfm command saves the \optsize to the \sizespec macro. The \optsize is in the form as shown below in the code where \regtfm is used. The \optfn \{fontname\} expands to the \fontname or to the corrected \fontname read from the \optsize macro at \reversetfm. It is used in the \fontlet macro.

The implementation detail: The \reg \fontid \{optical size data\} and all control sequences \reg \{fontname\} from this data line have the same meaning because of the \reversetfm macro. The \optfn expands this data line and apply \runoptfn. This macro selects the right result from the data line by testing with the current \optsize value.
Optical sizes data for preloaded 8-bit 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*.

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`.  

64
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 load. 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 2.12.2) 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, \famlet, \famvardef, \moddef, \_famdecl and def make local assignment.

2.13.5 The \fontdef declarator in detail

You can declare \(\text{(font-switch)}\) by the \fontdef command.

\[
\text{\fontdef}\{\text{\{font-switch\}}\}\{\text{\{family selector\}}\}\{\text{\{font modifiers\}}\}\{\text{\{variant selector\}}\}
\]

where \(\text{\{family selector\}}\) and \(\text{\{font modifiers\}}\) are optional and \(\text{\{variant selector\}}\) is mandatory.

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

The \fontdef does the following steps. It pushes the current font context to a stack, it does modifications of the font context by given \(\text{\{family selector\}}\) 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:

\[
\text{\famvardef}\{\text{\{new variant selector\}}\}\{\text{\{family selector\}}\}\{\text{\{font modifiers\}}\}\{\text{\{variant selector\}}\}
\]

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

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

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

\famvardef\mi \{\text{\{medium\}}\}
\famvardef\mi \{\text{\{medium\}}\}

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

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

When you are mixing fonts from more families then you probably run into a problem with incompatible ex-heights. This problem can be solved using \setfontsize and \famvardef macros.
\fontfam[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 forRoboto family and the original meaning of \mf is lost. But Op\TeX{} 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:

\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\}. \{\Heros\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:

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

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

Op\TeX{} defines similar internal commands \_ttfont, \_chapfont, \_secfont and \_seccfont, see section 2.26. The commands \_typosize and \boldify are used in these macros. They set the math fonts to given size too and they are defined in section 2.17.
2.13.9 Modifying font features

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

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

\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 OpTeX 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):

\setfontsize{⟨size spec⟩} % sets the font size
\setff{⟨font feature⟩} % adds the font feature
\setletterspace{⟨number⟩} % sets letter spacing
\setwordspace{⟨scaling⟩} % modifies word spacing

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

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

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

You can use \setff with other font features provided by LuaTeXX 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{colr=yes}\rm % if the font includes colored characters, use colors
\setff{upper}\rm % to uppercase (lower=lowecase) 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:
If you want to write such a font family file, you need to keep the following rules.

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

```
\famdecl [(Name of family)] \{Familyselector\} \{\{comments\}\}
\{\{modifiers\}\} \{\{variant selectors\}\} \{\{\fontnamegen\}\}
```

This writes information about font family at the terminal and prevents loading such file twice. Moreover, it probes existence of \fontnamegen in your system. If it doesn’t exist, the file loading is skipped with a warning on the terminal. The \famvardef macro must be defined in the last parameter of the \famdecl. More about it is documented below.

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

\famdecl...
{\def\fontnamegen\{xx-\_currV\}}

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\} 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 \( \langle \text{Familyselector} \rangle \) is defined by the \_famdecl macro as:

\protected\def\langle \text{Familyselector} \rangle {%
 \_def\_currfamily {\langle \text{Familyselector} \rangle}%
 \_def\_fontnamegen {\ldots}% 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 decoration. It runs the \( \langle \text{Familyselector} \rangle \) and it runs \_rm, so the first font from the new family is loaded and it is ready to use it.

Name conventions
Create font modifiers, new variants, and the \( \langle \text{Familyselector} \rangle \) 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 OpTiX warns about it. See the end of section 2.13.4.

The name of \( \langle \text{Familyselector} \rangle \) should begin with an uppercase letter.

Please, look at OpTiX 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, \kfi, \kif 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 OpTiX.

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

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

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

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

2.13.12 How to write the font family file with optical sizes
You can use \_optname macro when \_fontnamegen in expanded. This macro is fully expandable and its input is \( \langle \text{internal-template} \rangle \) and its output is a part of the font file name \( \langle \text{size-dependent-template} \rangle \) with respect to given optical size.

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

\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 \optsize \between 11.1 and 15 then 12 is substituted instead question mark. The \token{resizing-data} virtually begins with zero \nullchar, but it is not explicitly written. The right part of \token{resizing-data} must be terminated by * which means "less than infinity".

If \optname gets an argument which is not registered \token{internal-template} then it expands to \token{failedoptname} which typically ends with an error message about missing font. You can redefine \token{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 \token{resizing-data}. There are shortcuts \regoptFS, \regoptT, etc. here. The collection of \token{internal-templates} are declared, each of them covers a collection of real file names.

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

The \optname macro can be used even if no otical 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 \token{internal-templates} and do an alias from each \token{internal-template} to \token{real-font-name}. For example, you can do it as follows:

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

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

\begin{verbatim}
\__fdecl[Pol\_tawski]\Pol\_tawski \{Antykw a Półtawskiego, Polish traditional font family\}
\"\light \noexpd \expd \expd \cond \ccond \size \caps\} \{\rm \bf \it \bi\} \{
\{\antpolt-regular\}
\{\def\fontnamegen {\antpolt\_liV\_condV\_currV\_capsV\_fontfeatures}\}
\__u3\__detsizel\detokenize{\Modifiers:\n\light ...... light weight, \bf,\bi=semibold\\n\noexpd ...... no expanded, no condensed, designed for 10pt size (default)\\n\expd ...... expanded, designed for 6pt size\\n\expd ...... semi expanded, designed for 8pt size\\n\cond ...... semi condensed, designed for 12pt size\\n\ccond ...... condensed, designed for 17pt size\\n\psize ...... auto-sitches between \ccond \noexpd \expd \expd by size\\n\caps ...... caps & small caps\\n}\}
\__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 \expd \{\fsetV cond=semiexpd \}
\__moddef \cond \{\fsetV cond=semicond \}
\__moddef \ccond \{\fsetV cond=cond \}
\end{verbatim}

72
2.13.13 How to register the font family in the Font Selection System

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

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

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

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

```latex
\_famtext {Special name for printing a catalog :} \_faminfo [Catalogue] {Catalogue of all registered font families} {fonts-catalog} {} \_famalias [Catalog] \_famtext {Computer Modern like family :} \_faminfo [Latin Modern] {\TeX\ Gyre fonts based on Computer Modern} {f-lmfonts} { -, \caps: {\it} } \_famalias [LMfonts] [Latin Modern Fonts] \_famalias [lm] \_famtext {\TeX\ Gyre fonts based on Adobe 35 :} \_faminfo [Termes] {\TeX\ Gyre Termes fonts based on Times} {f-termes} { -, \caps, \cond, \caps\cond: {\it} } \_famalias [Termes] \_faminfo [Heros] {\TeX\ Gyre Heros fonts based on Helvetica} {f-heros} { -, \caps, \cond, \caps\cond: {\it} } \_famalias [Helvetcia] ...
```

The `\_faminfo` command has the syntax:

```latex
\_faminfo [{⟨Family Name⟩}] {⟨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 `⟨mods⟩ : {⟨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 \tt runs internal variant modifier \_fmodrm, \_fmodbf, \_fmodit, \_fmodbi and \_fmodtt. These modifiers save their data to the \_famv macro which is \rm or \bf or \it or \bi or \tt.

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

The \fontsel creates the ⟨font switch⟩ in the format \_ten⟨famv⟩ and loads the font associated to the ⟨font switch⟩. The loading is done by:

a) \letfont \langle font switch⟩ = \savedswitch \_sizespec
b) \font \langle font switch⟩ = \fontnamegen \_sizespec

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

After the font is loaded, final job is done by \_fontselA(⟨font-switch⟩).

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

The \newfontloaded should be defined for micro-typographic configuration of fonts, for example. The mte.opm package uses it. See also OpTeX trick 0058.

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

The \famdecl [{\fontnamegen{[data]}]} \{\fontnamegen{[data]}\}] runs \initunifonts, then checks if \famselector is defined. If it is true, then closes the file by \endinput. Else it defines \famselector and saves it to the internal \f: \{\fontnamegen{[data]}\}. The macro \initfontfamily uses 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 \famselector too, because \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 \{rm-template\} \{bf-template\} \{it-template\} \{bi-template\} saves data for usage by the \currv macro. If a template is only dot then previous template is used (it can be used if the font family doesn’t dispose with all standard variants).

\currv expands to a template declared by \fvars depending on the \{variant name\}. Usable only of standard four variants. Next variants can be declared by the \famvardef macro. \fsetv \{key}=\{value\}, \ldots, \{key}=\{value\} expands to \def\{key}\v\{value\} in the loop.
\onlyif \{key\}={value-a},\{value-b\}...,{\{value-z\}} \{what\} runs \{what\} only if the \{key\} is defined as \{value-a\} or \{value-b\} or ... or \{value-z\}.

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

\moddef \{modifier\} \{data\} simply speaking does \def\{modifier\}={\{data\}}, but we need to respect the family context. In fact, \protected\def\f:\{current family\}::\{modifier\}={\{data\}} is performed and the \{modifier\} is defined as \_famdepend\{modifier\}={\f::\currfamily::\{modifier\}}. It expands to \{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 \{cs-name\}=\_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 \{\data\} does:
\begingroup \data \ea \endgroup \the\font

It means that font modifiers used in \data are applied in the group and the resulting font (current at the end of the group) is set to the \font-switch. We want to declare \font-switch 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 \font-switch name.

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

but the macro \_xxx is declared as family-dependent. It is analogically as in \moddef. The \_xxx is defined as \_famdepend\xxx{f:\_currfamily:xxx} and \_f:{\famfamily}: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.

When the \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.

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 \famalias does
\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 \famtext 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 \_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.

\setletterspace is based on the special font features provided by luatexload 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.

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

\_optfontalias \{\{template A\}\} \{\{template B\}\} is defined as \let \_oz: \{template A\}=\_oz: \{template B\}.
2.14 Preloaded fonts for math mode

The Computer Modern and AMS fonts are preloaded here in classical math-fam concept, where each math family includes three fonts with max 256 characters (typically 128 characters).

On the other hand, when \fontfam macro is used in the document then text font family and appropriate math family is loaded with Unicode fonts, i.e. Unicode-math is used. It re-defines all settings given here.

The general rule of usage the math fonts in different sizes in OpTeX says: set three sizes by the macro \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.

\begin{verbatim}
\def\normalmath{% \loadmathfamily 0 cmr % CM Roman \loadmathfamily 1 cmmi % CM Math Italic \loadmathfamily 2 cmsy % CM Standard symbols \loadmathfamily 3 cmex % CM extra symbols \loadmathfamily 4 msam % AMS symbols A \loadmathfamily 5 msbm % AMS symbols B \loadmathfamily 6 rsfs % script \loadmathfamily 7 eufm % fractur \loadmathfamily 8 bbfsans % sans serif \loadmathfamily 9 bbisans % sans serif \setmathfamily 10 \tentt \setmathfamily 11 \tenbi \setmathdimens \}
\def\boldmath{% \loadmathfamily 0 cmbx % CM Roman Bold Extended \loadmathfamily 1 cmmib % CM Math Italic Bold \loadmathfamily 2 cmbsy % CM Standard symbols Bold \loadmathfamily 3 cmexb % CM extra symbols Bold \loadmathfamily 4 msam % AMS symbols A (bold not available?) \loadmathfamily 5 msbm % AMS symbols B (bold not available?) \loadmathfamily 6 reft % script (bold not available?) \loadmathfamily 7 eufb % fractur bold \loadmathfamily 8 bbfsans % sans serif extra bold \loadmathfamily 9 bbisans % sans serif extra bold \setmathfamily 10 \tentt \setmathfamily 11 \tenbi \setmathdimens \}
\end{verbatim}

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 \unimat-codes opm is loaded, see the section 2.16.2.
\_protected\def \_mit \{\_fam1 \}
\_protected\def \_cal \{\_fam2 \}
\_protected\def \_bbchar \{\_fam5 \} % double stroked letters
\_protected\def \_frak \{\_fam7 \} % fraktur
\_protected\def \_script \{\_fam6 \} % more extensive script than \_cal
\public \mit \cal \bbchar \frak \script ;

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

\_regtfm cmmi 0 cmmi5 5.5 cmmi6 6.5 cmmi7 7.5 cmmi8 8.5 cmmi9 9.5
cmmi10 11.1 cmmi12 *
\_regtfm cmmib 0 cmmib5 5.5 cmmib6 6.5 cmmib7 7.5 cmmib8 8.5 cmmib9 9.5 cmmib10 *
cmmib12 *
\_regtfm cmtex 0 cstex8 8.5 cstex9 9.5 cstex10 *
\_regtfm cmasy 0 cmasy5 5.5 cmasy6 6.5 cmasy7 7.5 cmasy8 8.5 cmasy9 9.5 cmasy10 *
\_regtfm cmbx 0 cmbx5 5.5 cmbx6 6.5 cmbx7 7.5 cmbx8 8.5 cmbx9 9.5 cmbx10 *
\_regtfm cmex 0 cmex8 8.5 cmex9 9.5 cmex10 *
\_regtfm cmexb 0 cmexb10 *
\_regtfm cmr 0 cmr5 5.5 cmr6 6.5 cmr7 7.5 cmr8 8.5 cmr9 9.5 cmr10 11.1 cmr12 15 cmr17 *
\_regtfm cmbx 0 cmbx5 5.5 cmbx6 6.5 cmbx7 7.5 cmbx8 8.5 cmbx9 9.5
cmbx10 11.1 cmbx12 *
\_regtfm cmti 0 cmti7 7.5 cmti8 8.5 cmti9 9.5 cmti10 11.1 cmti12 *
\_regtfm cmtt 0 cmtt8 8.5 cmtt9 9.5 cmtt10 11.1 cmtt12 *

%% AMS math fonts, optical sizes:
\_regtfm msam 0 msam5 5.5 msam6 6.5 msam7 7.5 msam8 8.5 msam9 9.5 msam10 *
\_regtfm msbm 0 msbm5 5.5 msbm6 6.5 msbm7 7.5 msbm8 8.5 msbm9 9.5 msbm10 *

%% fraktur, rsfs, optical sizes:
\_regtfm eufm 0 eufm5 6 eufm7 8.5 eufm10 *
\_regtfm eufb 0 eufb5 6 eufb7 8.5 eufb10 *
\_regtfm rsfs 0 rsfs5 6 rsfs7 8.5 rsfs10 *

%% bf and bi sansserif math alternatives:
\_regtfm bfsans 0 ecssx0500 5.5 ecssx0600 6.5 ecssx0700 7.5 ecssx0800
8.5 ecssx0900 9.5 ecssx1000 11.1 ecssx1200 *
\_regtfm bbsans 0 ecssx0500 5.5 ecssx0600 6.5 ecssx0700 7.5 ecssx0800
8.5 ecssx0900 9.5 ecssx1000 11.1 ecssx1200 *
\_regtfm bbfsans 0 ecssx0500 5.5 ecssx0600 6.5 ecssx0700 7.5 ecssx0800
8.5 ecssx0900 9.5 ecssx1000 11.1 ecssx1200 *

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

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

The \_corrmsize \langle factor \rangle \langle space \rangle can be used just before \_loadmathfamily or \_setmathfamily. The \langle factor \rangle is decimal number, it denotes scale-factor “size of loaded math font in \textstyle : size of text font”. You can use it in \normalmath or \boldmath macros if you want to do a corrections (for example due to different ex-height in text and math font). The \_corrmsize is applied only to one following \_loadmathfamily or \_setmathfamily. If it is missing then the \langle factor \rangle is 1 for such math family (i.e. no size corrections).
The \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\Umathspaceafterscript is used instead \scriptspace setting because Lua\TeX ignores \scriptspace in most cases. There is small difference from classical \TeX: we set “scaled” \Umathspaceafterscript dependent on textstyle, scriptstyle, etc. sizes. The \scriptspacefactor is set to 0.05 which gives the same result as Plain \TeX \scriptspace=0.5pt 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.

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

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

The Lua code presented here solves this problem. But you cannot set your own control sequence in the form \texttt{\langle word \rangle} \texttt{\_} or \texttt{\langle word \rangle \_\langle one-letter \rangle} (where \texttt{\langle word \rangle} is a sequence of letters) because such control sequences are inaccessible: preprocessor rewrites it.

The \texttt{\mathsbon} macro activates the rewriting rule \texttt{\langle word \rangle \_\langle nonletter \rangle} to \texttt{\langle word \rangle \_\langle letter \rangle\langle nonletter \rangle} and \texttt{\langle word \rangle \_\langle letter \rangle\langle nonletter \rangle} to \texttt{\langle word \rangle \_\langle letter \rangle\langle nonletter \rangle} at input processor level. The \texttt{\mathsboff} deactivates it. You can ask by \texttt{\_ifmathsb} if this feature is activated or deactivated. By default, it is activated in the \texttt{\everyjob}, see section 2.1. Note, that the \texttt{\everyjob} is processed after the first line of the document is read, so the \texttt{\mathsbon} is activated from the second line of the document.

All mathcodes are set to equal values as in plain\TeX. But all encoding-dependent declarations (like these) will be set to different values when a Unicode-math font is used.
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 OpTeX 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 \texttt{protected} in Op\TeX.
These macros are defined similarly as in plainTeX. Only internal macro names from plainTeX with @ character are re-written in a more readable form.

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

Active \prime character is defined here.

Active \texttt{Z} character is like \texttt{ne} in math % obsolete

\texttt{\scalebig} macro is based on the \texttt{\scalebig} macro because we need the dependency on the various sizes of the fonts.
protected\def\Bigl{\mathopen\Big}\protect\def\Bigr{\mathclose\Big}\protect\def\Bigm{\mathrel\Big}\protected\def\bigl{\mathopen\bigg}\protect\def\bigr{\mathclose\bigg}\protect\def\bigm{\mathrel\bigg};

Math relations defined by the \jointrel plain \TeX macro:

\protected\def\joinrel{\mathrel{\mkern-2.5mu}} % -3mu in plain\TeX
\protected\def\Relbar{\mathrel{=}}
\mathchardef\lhook=\textasciitilde\textasciitilde\textasciitilde % \_smash, because - has the same height as +
\protected\def\hookrightarrow{\lhook\joinrel\rightarrow}
\mathchardef\rhook=\textasciitilde\textasciitilde\textasciitilde
\protected\def\hookleftarrow{\leftarrow\joinrel\rhook}
\protected\def\bowtie{\mathrel{\triangleright\joinrel\mathrel{\triangleleft}}}
\protected\def\models{\mid\joinrel=}
\protected\def\Longrightarrow{\Relbar\joinrel\Rightarrow}
\protected\def\longrightarrow{\relbar\joinrel\rightarrow}
\protected\def\longleftarrow{\leftarrow\joinrel\relbar}
\protected\def\Longleftarrow{\Leftarrow\joinrel\Relbar}
\protected\def\longmapsto{\mapstochar\longrightarrow}
\protected\def\longleftrightarrow{\leftarrow\joinrel\rightarrow}
\protected\def\Longleftrightarrow{\Leftarrow\joinrel\Rightarrow}
\protected\def\iff{\thickapprox\Longleftrightarrow\thickapprox}

\protected\def\ldots{\mathinner{\ldotp\ldotp\ldotp}}
\protected\def\cdots{\mathinner{\cdotp\cdotp\cdotp}}
\protected\def\vdots{\vbox{\baselineskip=.4em \lineskiplimit=\zo \kern.6em \hbox{.}\hbox{.}\hbox{.}}}
\protected\def\ddots{\mathinner{\mkern1mu\raise.7em\vbox{\kern.7em\hbox{.}}\mkern2mu\raise.4em\hbox{.}\mkern2mu\raise.1em\hbox{.}\mkern1mu}}
\public\joinrel;
\ldots\cdots\vdots\ddots;

\adots inspired by plain \TeX
\public\adots;

Math accents (encoding dependent declarations).
\math, \mathaccent, \skew, \overrightarrow, \overleftarrow, \overbrace, \underbrace macros. The last four are redefined when Unicode math is loaded.

Macros based on \delimiter, \*witdelims and \radical primitives.
\texttt{\def\rootA#1#2{\_setbox0=\_hbox{$\_math#1\sqrt{#2}$}\_dimen0=\_ht0\_advance\_dimen0by-\_dp0\_mkern5mu\_raise.6\_dimen0\_copy\_rootbox\_mkern-10mu\_box0}}

\texttt{\_newifi\_ifvp \_newifi\_ifhp}

\texttt{\_protected\_def\_vphantom{\_vptrue\_hpfalse\_phant}}

\texttt{\_protected\_def\_hphantom{\_vpfalse\_hptrue\_phant}}

\texttt{\_protected\_def\_phantom{\_vptrue\_hptrue\_phant}}

\texttt{\_def\phantom{\ifmmode\_def\_next{\_mathpalette\_mathphant}%\else\_let\_next=\_makephant\fi\_next}}

\texttt{\_def\_makephant#1{\_setbox0=\_hbox{#1}\_finphant}}

\texttt{\_def\_mathphant#1#2{\_setbox0=\_hbox{$\_math#1\{#2\}$}\_finphant}}

\texttt{\_def\_finphant{\_setbox2=\_null\_ifvp\_ht2=\_ht0\_dp2=\_dp0\_fi\_ifhp\_wd2=\_wd0\_fi\_box0}}

\texttt{\_def\_mathstrut{\_vphantom(}}

\texttt{\_protected\_def\_smash{\_relax % \_relax, in case this comes first in \halign}}

\texttt{\_ifmmode\_def\_next{\_mathpalette\_mathsmash}\else\_let\_next=\_makesmash\fi\_next}}

\texttt{\_def\_makesmash#1{\_setbox0=\_hbox{#1}\_finsmash}}

\texttt{\_def\_mathsmash#1#2{\_setbox0=\_hbox{$\_math#1\{#2\}$}\_finsmash}}

\texttt{\_def\_finsmash{\_ht0=\_zo\_dp0=\_zo\_hbox{\_box0}}}
The \cases and \bordermatrix macros are almost identical as in plain \TeX. You can simply re-define \bordermatrix with other delimiters using the common \_bordermatrixwithdelims macro.

The \eqalign macro behaves like in Plain \TeX by default. It creates the \vcenter 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 Opt\TeX, this macro is more flexible. See section 4.4 in the Typesetting Math with Opt\TeX. 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): rlclrlcrlc etc. where \r and \l columns are without spaces and \c column (if used) has space \eqspace/2 at its both sides.

The \displaylines{⟨formula⟩\cr⟨formula⟩\cr...⟨formula⟩} 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.

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

\openup, \equalignno and \leqalignno macros are copied from Plain \TeX unchanged.
These macros are inspired by ams-math.tex file.

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

\[
\texttt{not< becomes \_nless} \\
\texttt{not> becomes \_ngtr} \\
\texttt{if \_notXXX is defined, \texttt{not}XXX becomes \_notXXX;} \\
\texttt{if \_nXXX is defined, \texttt{not}XXX becomes \_nXXX;} \\
\texttt{otherwise, \texttt{not}XXX is done in the usual way.}
\]
\currstyle which expands to displaystyle, textstyle, scriptstyle or scriptscriptstyle depending on the current math style when \mathstyles was opened.

\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 \stylenum is 0, 1, 2 or 3 depending on the current math style when \mathstyles was opened.

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

\math-macros.opm
\newcount\stylenum
\def\mathstyles#1{{\mathchoice{\stylenum0 #1}{\stylenum1 #1}1\stylenum2 \stylenum3 #1}}
\def\dobystyle#1#2#3#4{\ifcase\stylenum#1\or#2\or#3\or#4\fi}
\def\currstyle{\dobystyle displaystyle textstyle scriptstyle scriptscriptstyle}
\public \mathstyles \dobystyle \currstyle \stylenum ;

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

Second note: \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:
$\cramped {x^2\over y^2}$

\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.

\math-macros.opm
\def\mathbox#1{{\mathstyles{\hbox{\ifnum\stylenum<2 \everymath{\currstyle}\else\ifnum\stylenum=2 \def\textmff{ssty=1;}\fi\ifnum\stylenum=3 \def\textmff{ssty=2;}\def\scriptmff{ssty=2;}\fi\typoscale\dobystyle{}{}{700}{500}/\fi #1}}}}
\public \mathbox ;

2.16 Unicode-math fonts

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

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

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

The main math font is loaded by \loadmath (typically indirectly using \fontfam) and you can load more additional math fonts by \addUmathfont:
\addUmathfont \famname {⟨{⟨normal-font⟩}⟨{⟨ffeatures⟩} {⟨{⟨bold-font⟩}⟨{⟨ffeatures⟩} {⟨{factor⟩}}}}}}

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. Example:
\fontfam[lm] % does \lodmath[[latinmodern-math]] 
\addUmathfont \xits {{XITSMath-Regular}}{{XITSMath-Bold}} {} {}
declares latinmodern-math as main math font (its bold variant is “faked bold”) and the additional math font \xits is XITSMath-Regular for normal printing and XITSMath-Bold for bold printing.

All characters used in math formula is printed from main math font by default. But you can redeclare a list of characters for printing from additional font by \resetmathchars\famname {list of sequences} ;. For example:

\resetmathchars \xits \stareq \triangleq \veeeq \wedgeq ;
sets the characters \stareq, \triangleq, \veeeq, \wedgeq from the \xits additional font. You can see all possible sequences by \input print-unimath.opm. If you want to use whole math alphabet from additional math font, use \addto\selector{\fam{\famname}}, for example \addto\cal{\fam{xits}} declares all \cal characters from the \xits font loaded by \addUmathfont.

2.16.1 Unicode-math macros preloaded in the 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 Lua\LaTeX\ 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 with 8bit Math font at family 4. See definition of \script macro where \fam4 is used.

\begin{verbatim}
\_def\_normalunimath{%
\_loadumathfamily 1 {\_unimathfont}{} % Base font
\_loadmathfamily 4 rsfs % script
\_setunimathdimens
}%
\_def\_boldunimath{%
\_ifx\_unimathboldfont \_undefined
\_loadumathfamily 1 {\_unimathfont}{embolden=1.7;} % 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{\_everymath{}$\_fam1\_displaystyle{0\_atop0}$}%
\_Umathfractiondelsize\_displaystyle = \_dimexpr(\_ht0-\_Umathaxis\_displaystyle)*2\_relax
\_setbox0=\_box\_voidbox}
\end{verbatim}

If you try the example above about \loadboldmath{[xitsmath-bold]} \to {[xitsmath-regular]} then you can find a bug in XITSMath-Bold font: the symbols for norm ‖𝑥‖ are missing. So, we have to define \_boldmath macro manually. The missing symbol is loaded from family 5 as no-bold variant in our example:

\begin{verbatim}
\\\loadmath{[xitsmath-regular]}
\_def\_boldmath{%
    \_loadumathfamily 1 {[xitsmath-bold]}{} % Base font
    \_loadmathfamily 4 rsfs % script
    \_loadumathfamily 5 {[xitsmath-regular]}{}
    \_def\{|{\_Udelimiter 0 5 "02016 }% % norm delimiter from family 5
    \_setmathdimens
}\end{verbatim}
Unicode math font includes all typical math alphabets together, user needs not to load more \TeX math families. These math alphabets are encoded by different parts of Unicode table. We need auxiliary macros for setting mathcodes by selected math alphabet.\textbackslash \umathrange \{\langle from\rangle\textendash\langle to\rangle\} \\langle class\rangle \langle family\rangle \langle first\rangle sets \textbackslash \umathcodes of the characters in the interval \langle from\rangle\textendash\langle to\rangle to \langle first\rangle, \langle first\rangle+1, \langle first\rangle+2 etc., but \textbackslash \umathcharholes are skipped. \textbackslash \umathcharholes of the Unicode table not designed for math alphabets, they cause that the math alphabets are not continuously spread out in the table; I mean that the designers were under the influence of drugs when they created this part of the Unicode table. The \langle from\rangle\textendash\langle to\rangle clause includes characters like A\textendash Z. \textbackslash \umathrangeGREEK \langle first\rangle is the same as \textbackslash \umathrange \langle (alpha)\textendash(omega) \rangle \langle first\rangle. \textbackslash \umathrangeGREEK \langle first\rangle is the same as \textbackslash \umathrange \langle (Alpha)\textendash(Omega) \rangle \langle first\rangle. \textbackslash \greekdef \langle control sequences \rangle \\textbackslash \relax defines each control sequence as a normal character with codes \textbackslash \umathnumB, \textbackslash \umathnumB+1, \textbackslash \umathnumB+2 etc. It is used for redefining the control sequences for math Greek \textbackslash \alpha, \textbackslash \beta, \textbackslash \gamma etc.
\resetmathchars \{fam\} \{list of sequences\} ; saves \{fam\} to \_mafam and applies \_resetmathcharsX to each sequence from \{list of sequences\}. The \_directlua chunk prints the 8-digits hexadecimal code of the sequence followed by the sequence itself. The digits are scanned by \_resetmathcharsY and new \_Umathchardef and \_Umathcode is declared. Note that the used primitives have the syntax:

\[ \Umathchardef \{sequence\} \{math class\} \{math family\} \{slot number\} \]
\[ \_Umathcode \{code\} \{math class\} \{math family\} \{slot number\} \]

\begin{verbatim}
\_def \_resetmathchars #1{\_chardef \_mafam=#1 \_relax \_xargs \_resetmathcharX}
\_def \_resetmathcharX#1{\_ea \_resetmathcharY \_directlua{tex.print(string.format("\_pcent08X", \_the#1))}#1}
\_def \_resetmathcharY#1#2#3#4#5#6#7#8#9{% \_9 is given \_math-sequence
\_Umathchardef #9\_numexpr #3/2 \_relax \_mafam #4#5#6#7#8
\_Umathcode #4#5#6#7#8\_numexpr #3/2 \_relax \_mafam #4#5#6#7#8
}
\_public \addUmathfont \resetmathchars ;
\end{verbatim}

\subsection{2.16.2 Macros and codes set when \loadmath is processed firstly}

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

\begin{verbatim}
\_codedecl \_ncharrmA \(\text{Uni math codes <2022-07-20>}\) \% preloaded on demand by \loadmath
\\_umathnumB="0391
\_greekdef \Alpha \Beta \Gamma \Delta \Epsilon \Zeta \Eta \Theta \Iota \Kappa
\Lambda \Mu \Nu \Xi \Omicron \Pi \upsilon \Phi \Chi \Psi \Omega \relax
\_umathnumB="03B1
\_greekdef \alpha \beta \gamma \delta \varepsilon \zeta \eta \theta \iota \kappa
\lambda \mu \nu \xi \omicron \pi \upsilon \phi \chi \psi \omega \vardelta \epsilon \vartheta \varkappa \phi \relax
\_protected \_def \_rmvariables \{ \umathrange{A-Z}\_ncharrmA \umathrange{(a-z)}\_ncharrma\}
\_protected \_def \_bfvariables \{ \umathrange{A-Z}\_ncharrbA \umathrange{(a-z)}\_ncharrba\}
\_protected \_def \_itvariables \{ \umathrange{A-Z}\_ncharitA \umathrange{(a-z)}\_ncharita\}
\_protected \_def \_bivariables \{ \umathrange{A-Z}\_ncharbiA \umathrange{(a-z)}\_ncharbia\}
\_protected \_def \_calvariables \{ \umathrange{A-Z}\_ncharclA \umathrange{(a-z)}\_ncharcla\}
\_protected \_def \_calvariables \{ \umathrange{(A-Z)}\_ncharrA \umathrange{(a-z)}\_ncharrm\}
\end{verbatim}

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

\_chardef\_digitrmO=`0 \_chardef\_digitbfO=1D7CE
\_chardef\_digitbbO=1D7D8 \_chardef\_digitsnO=1D7E2
\_chardef\_digitbsO=1D7EC

\_protected\_def\_rmdigits { \_umathrange{0-9} \_digitrmO}
\_protected\_def\_bfdigits { \_umathrange{0-9} \_digitbfO}
\_protected\_def\_bbdigits { \_umathrange{0-9} \_digitbbO}
\_protected\_def\_sansdigits { \_umathrange{0-9} \_digitsnO}
\_protected\_def\_bsansdigits { \_umathrange{0-9} \_digitbsO}
\_protected\_def\_ttdigits { \_umathrange{0-9} \_digitttO}

The math alphabets \cal, \bbchar, \frak, \script are re-defined here. The \_marm, \_mabf, \_mait, \_mabi, \_matt used in \rm, \bf, \it, \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:

\_protected\_def\_mabf {\_inmath{\_bfvariables\_bfgreek\_bfGreek\_bfdigits}}
\_protected\_def\_mabi {\_inmath{\_bivariables\_bigreek\_bfGreek\_bfdigits}}
\_inmath{\_cmds} applies \_cmds only in math mode.
You can redefine these macros to follow your wishes. For example, you need upright lowercase Greek letters, you don't need \bf and \bi behave as sans serif in math, ...

\protected\def\marm{\_inmath{\_rmvariables \_rmdigits}}
\protected\def\mait{\_inmath{\_itvariables \_itGreek}}
\protected\def\mabf{\_inmath{\_bsansvariables \_bsansgreek \_bsansGreek \_bsansdigits}}
\protected\def\mabi{\_inmath{\_bisansvariables \_bisansgreek \_bsansGreek \_bsansdigits}}
\protected\def\matt{\_inmath{\_ttvariables \_ttdigits}}
\protected\def\bbchar{\_bbvariables \_bbdigits}
\protected\def\frak{\_frakvariables}
\protected\def\misans{\_isansvariables \_sansdigits}
\protected\def\mbisans{\_bisansvariables \_bisansgreek \_bsansGreek \_bsansdigits}
\protected\def\script{\_rmvariables \_fam4}
\protected\def\mit{\_itvariables \_rmdigits \_itGreek \_rmGreek}

\public \bbchar \cal \frak \misans \mbisans \script \mit;

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

\begingroup % \input mathclass.opm (which is a copy of MathClass.txt):
\long\def p#1;#2 {\_ifx^#2^ \_else \edef \tmp {\_csname _c:#2 \endcsname} \_if \relax \tmp \_else \pA#1....\end#2 \fi \_ea \p \_fi }
\def pA#1..#2..#3\end#4{%\_ifx \_relax #2 \relax \_pset{"#1}{#4} \_else \_fornum "#1.."#2 \do{\_pset{##1}{#4}} \fi }
\sdef{_c:L}{1}\sdef{_c:B}{2}\sdef{_c:V}{2}\sdef{_c:R}{3}\sdef{_c:N}{0}\sdef{_c:U}{0}
\sdef{_c:F}{0}\sdef{_c:O}{4}\sdef{_c:C}{5}\sdef{_c:P}{6}\sdef{_c:A}{7}
\def pset#1#2{\Umathcode#1=\tmp \space 1 #1 \relax \_if#2O \Udelcode#1=1 \_else \_pset{\_c:#2}{#1} \fi \_if#2C \Udelcode#1=1 \_else \_pset{\_c:#2}{#1} \fi \_if#2F \Udelcode#1=1 \_else \_pset{\_c:#2}{#1} \fi \_if#2\_Udelcode#1=1 \_else \_pset{\_c:#2}{#1} \fi \_if#2G \Udelcode#1=1 \_else \_pset{\_c:#2}{#1} \fi \_if#2T \Udelcode#1=1 \_else \_pset{\_c:#2}{#1} \fi \_if#2\_Udelcode#1=1 \_else \_pset{\_c:#2}{#1} \fi \_if#2\_Udelcode#1=1 \_else \_pset{\_c:#2}{#1} \fi }
\catcode`#=14 \everyeof={;{} }
\def\par{}\globaldefs=1 \_ea \p \_input mathclass.opm
\endgroup

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

\UnicodeMathSymbol{00393}{\mupGamma}{\mathalpha}{capital gamma, greek}
\UnicodeMathSymbol{00394}{\mupDelta}{\mathalpha}{capital delta, greek}
\UnicodeMathSymbol{00395}{\mupEpsilon}{\mathalpha}{capital epsilon, greek}
\UnicodeMathSymbol{00396}{\mupZeta}{\mathalpha}{capital zeta, greek}
\UnicodeMathSymbol{00397}{\mupEta}{\mathalpha}{capital eta, greek}
\UnicodeMathSymbol{00398}{\mupTheta}{\mathalpha}{capital theta, greek}
\UnicodeMathSymbol{00399}{\mupIota}{\mathalpha}{capital iota, greek}
\UnicodeMathSymbol{0039A}{\mupKappa}{\mathalpha}{capital kappa, greek}

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: unimath-table.opm

\UnicodeMathSymbol{00393}{\mupGamma}{\mathalpha}{capital gamma, greek}
\UnicodeMathSymbol{00394}{\mupDelta}{\mathalpha}{capital delta, greek}
\UnicodeMathSymbol{00395}{\mupEpsilon}{\mathalpha}{capital epsilon, greek}
\UnicodeMathSymbol{00396}{\mupZeta}{\mathalpha}{capital zeta, greek}
\UnicodeMathSymbol{00397}{\mupEta}{\mathalpha}{capital eta, greek}
\UnicodeMathSymbol{00398}{\mupTheta}{\mathalpha}{capital theta, greek}
\UnicodeMathSymbol{00399}{\mupIota}{\mathalpha}{capital iota, greek}
\UnicodeMathSymbol{0039A}{\mupKappa}{\mathalpha}{capital kappa, greek}
We have to read this information and convert it to the Unicode math codes.

Many special characters must be declared with care...

\global\Udelcode`\leq\equal\ GE1 "027E8 % these characters have different meaning
\global\Udelcode`\geq\equal\ GE1 "027E9 % as normal and as delimiter
% hyphen character is transformed to minus:
\Umathcode`\=-2 1 "2212
% mathclass defines : as Punct, plain.tex as Rel, we keep mathclass,
% i.e. there is difference from plain.tex, you can use $A:B$ to $\frac{A}{B}$.
% mathclas defines ! as Ord, plain.tex as Close
\Umathcode`\!=5 1 `!
\Umathchardef\mathexclam=5 1 `!
% mathclass defines ? as Punct, plain.tex as Close
\Umathcode`\?=5 1 `?
\Umathchardef\mathquestion=5 1 `?
% mathclass defines * as Ord, plain.tex as Bin
\Umathcode`\ast=2 1 "02217 % equivalent to \ast, like in plain TeX
\Umathcode`\in=0 1 \in % usage $\infty$ can be seen in old documents
\protected\def\_sqrt{\Uradical1 "0221A}
\protected\def\_cuberoot{\Uradical1 "0221B}
\protected\def\_fourthroot{\Uradical1 "0221C}
\nabla{\textasciitilde} % \nabla behaves as uppercase Gereek letter, see \_setnabla
\sqrt\cuberoot\fourthroot;
\public\sqrt\cuberoot\fourthroot;
\public\_sqrt\_cuberoot\_fourthroot;
\\def\_intwithnolimits#1#2\lbrace\_ifx\#1\relax\_else
\ea\_let\_csname\_csstring\#1op\_endcsname\#1\rbrace
\_ifx\#1\mathopen\_def\#2\__Udelimiter4 1 \#1\_fi
\_ifx\#1\mathclose\_def\#2\__Udelimiter5 1 \#1\_fi
\_ifx\#1\mathaccent\_def\#2\__Umathaccentfixed7 1 \#1\_fi
\global\def\_globaldefs1\_input\unimath-table.ope
\endgroup

98
Aliases are declared here. They are names not mentioned in the `unimath-table.opm` file but commonly used in TeX.
\_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 \hksearow=\hksearrow
\_let \square=\mdlgwhtsquare
\_let \upalpha=\mupalpha
\_let \upbeta=\mupbeta
\_let \upgamma=\mupgamma
\_let \updelta=\mupdelta
\_let \upepsilon=\mupvarepsilon
\_let \uppi=\muppi
\_let \upvarpi=\mupvarpi
\_let \upchi=\mupchi
\_let \uppsi=\muppsi
\_let \upomega=\mupomega
The \_not macro is redefined here. If the \_not!\_csstring{char} is defined (by \_negationof) then this macro is used. Else centered / is printed over the \_csstring{char}.

\_protected\_def \_not!#1{% 
\_trycs{\_not!\_csstring{#1}}{\_mathrel{\_mathstyles{ 
\_setbox0=\_hbox{\_math{\_currstyle{#1}}} 
\_hbox to \wd0{\_hss{\_currstyle{/}}\_hss} \_kern-\wd0 \_box0}}}%
\_negationof = \neq
\_negationof < \nless
\_negationof > \ngtr
\_negationof \_gets=\nleftarrow
\_negationof \_noteq=\nsimeq
\_negationof \_equal=\ne
\_negationof \_le=\nleq
\_negationof \_ge=\ngeq

100
Newly declared public control sequences are used in internal macros by OpTEX. We need to get new meanings for these control sequences in the private namespace.

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 opi. The big table with all math symbols is printed.
2.17 Scaling fonts in document (high-level macros)

These macros are documented in section 1.3.2 from the user point of view.
\texttt{\textbackslash typoscale \{(font-factor)/(baseline-factor)\}} scales font size and baselineskip by given factors in respect to current values. It calculates the \texttt{\textbackslash typosize} parameters and runs the \texttt{\textbackslash typosize}.

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

- 0: no math fonts are loaded. User must use \texttt{\textbackslash normalmath} or \texttt{\textbackslash boldmath} explicitly.
- 1: \texttt{\textbackslash normalmath} is run if \texttt{\textbackslash typosize/\textbackslash typoscale} are used first or they are run at outer group level. No \texttt{\textbackslash everymath/\textbackslash everydisplay} are set in this case. If \texttt{\textbackslash typosize/\textbackslash typoscale} are run repeatedly in a group then \texttt{\textbackslash normalmath} is run only when math formula occurs. This is done using \texttt{\textbackslash everymath/\textbackslash everydisplay} and \texttt{\textbackslash setmathfonts. mfontsrule=1} is default.
- 2: \texttt{\textbackslash normalmath} is run whenever \texttt{\textbackslash typosize/\textbackslash typoscale} are used. \texttt{\textbackslash everymath/\textbackslash everydisplay} registers are untouched.
\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}
        \let\runboldmath=\relax \fi
    \or \normalmath \fi}
\def\scalemain {%
  \ifdim\mainfsize=\zo
    \mainfsize=10pt \mainbaselineskip=12pt
    \let\setmainvalues=\setmainvaluesL
  \fi
  \optsize=\mainfsize \baselineskip=\mainbaselineskip
}
\public \scalemain \mainfsize \mainbaselineskip \mfontsrule ;

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

\texttt{\textbackslash thefonsize [(size)] and \textbackslash thefontscale [(factor)] do modification of the size of the current font. They are implemented by the \texttt{\newcurrforsize} macro.}

\em keeps the weight of the current variant and switches roman ↔ italic. It adds the italic correction by the \texttt{\additcorr} and \texttt{\afteritcorr} macros. The second does not add italic correction if the next character is dot or comma.

The \texttt{\boldify} macro does \texttt{\let\rm\bf}, \texttt{\let\it\bi} and \texttt{\let\normalmath=\boldmath}. All following text will be in bold. If should be used after \texttt{\textbackslash typosize} or \texttt{\textbackslash typoscale} macros. The internal \texttt{\runboldmath} macro runs \texttt{\boldmath} immediately if no delay of the math font loading is set by \texttt{\setmainvaluesL}.

The \texttt{\rm}, \texttt{\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 \_\textit{rmfixed} macro. It sets the \textit{rm} font from the default font size (declared by first \texttt{\textbackslash{typo}size} command and redefines itself be only the font switch for the next pages.

\begin{lstlisting}[language=TeX]
\def\_\textit{rmfixed}{% used in default \texttt{\textbackslash{footline}}
\ifdim\mainfotosize=0pt \mainfotosize=10pt \fi
\fontdef\tenrm{\resetmod\rm}
\global\let\_\textit{rmfixed}={\tenrm}% next use will be font switch only
\_\textit{rmfixed}
\}
\let\rmfixed=\tenrm % user can redefine it
\end{lstlisting}

### 2.18 Output routine

The output routine \_\texttt{\textbackslash{optexoutput}} is similar as in plain \LaTeX. It does:

- \_\texttt{\textbackslash{begoutput}} which does:
  - increments \textit{\textbackslash{pageno}},
  - prints \_\texttt{\textbackslash{Xpage}}{\langle \textit{\textbackslash{pageno}} \rangle} to the .ref file (if \texttt{\textbackslash{openref}} is active),
  - calculates \textit{\textbackslash{hoffset}},
  - sets local meaning of macros used in headlines/footlines (see \_\texttt{\textbackslash{regmacro}}).
- \_\texttt{\textbackslash{shipout}\_\texttt{\textbackslash{completepage}}}, which is \texttt{\textbackslash{vbox}} of –
  - background box, if \textit{\textbackslash{pgbackground}} is non-empty,
  - headline box by \_\texttt{\textbackslash{makeheadline}}, if the \_\texttt{\textbackslash{headline}} is nonempty,
  - \texttt{\textbackslash{vbox}} to\texttt{\textbackslash{vsize}} of \_\texttt{\textbackslash{pagecontents}} which consists of –
    - \_\texttt{\textbackslash{pagedest}}, the page destination pg:{\langle \textit{\textbackslash{pageno}} \rangle} for hyperlinks is created here,
    - \texttt{\textbackslash{topins}} box if non-empty (from \texttt{\textbackslash{topinserts}}),
    - \_\texttt{\textbackslash{box255}} with completed vertical material from main vertical mode,
    - \_\texttt{\textbackslash{footnoterule}} and \texttt{\textbackslash{footins}} box if nonempty (from \texttt{\textbackslash{fnote}}, \texttt{\textbackslash{footnote}}),
    - \texttt{\textbackslash{pgbottomskip}} (default is 0pt).
  - footline box by \_\texttt{\textbackslash{makefootline}}, if the \_\texttt{\textbackslash{footline}} is nonempty
- \_\texttt{\textbackslash{endoutput}} which does:
  - increments \_\texttt{\textbackslash{pageno}} using \texttt{\textbackslash{advancepageno}}
  - runs output routine repeatedly if \texttt{\textbackslash{dosupereject}} is activated.

\begin{lstlisting}[language=TeX]
\_\texttt{\textbackslash{begoutput}}\_\texttt{\textbackslash{endoutput}}
\end{lstlisting}

\_\texttt{\textbackslash{optexoutput}} is the default output routine. You can create another...

The \_\texttt{\textbackslash{preshipout}} (destination box number)/(box specification) used here behaves similarly like \texttt{\textbackslash{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.

\begin{lstlisting}[language=TeX]
\_\texttt{\textbackslash{preshipout}}\_\texttt{\textbackslash{\textbackslash{endoutput}}}
\end{lstlisting}

Default \_\texttt{\textbackslash{begoutput}} and \_\texttt{\textbackslash{endoutput}} is defined. If you need another functionality implemented in the output routine, you can \_\texttt{\textbackslash{addto\_\textbackslash{begoutput}}\_\texttt{\textbackslash{\ldots}}} or \_\texttt{\textbackslash{addto\_\textbackslash{endoutput}}\_\texttt{\textbackslash{\ldots}}} . The settings here are local in the \texttt{\textbackslash{output}} group.

The \_\texttt{\textbackslash{prepoffsets}} can set \texttt{\textbackslash{hoffset}} differently for the left or right page. It is re-defined by the \_\texttt{\textbackslash{margins}} macro.

The \_\texttt{\textbackslash{regmark}} tokens list includes accumulated \_\texttt{#2} from the \_\texttt{\textbackslash{regmacro}}. Logos and other macros are re-defined here (locally) for their usage in headlines or footlines.
The \texttt{\_xhsize} value can be changed at various places in the document but we need to have a constant \texttt{\_xhsize} in the output routine (for headlines and footlines, for instance). This value is set from the current value of \texttt{\hsize} when \texttt{\_setxhsize} macro is called. This macro destroys itself, so the value is set only once. Typically it is done in \texttt{\ margins} macro or when first \texttt{\ optexoutput} routine is called (see \texttt{\_begoutput}). Or it is called at the beginning of the \texttt{\begtt...\endtt} environment before \texttt{\hsize} value is eventually changed by the user in this environment.

\texttt{\_setxhsize} \texttt{(\_global\_xhsize=\_hsize \_global\_let\_setxhsize=\_relax)}

\texttt{\_completepage} is similar to what plain \TeX{} does in its output routine. New is only \texttt{\_backgroundbox}. It is \texttt{\vbox} with zero height with its contents (from \texttt{\pgbackground}) extended down. It is shifted directly to the left-upper corner of the paper.

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

\texttt{\_makeheadline} creates \texttt{\vbox} to\texttt{Opt} with its contents (the \texttt{\headline}) shifted by \texttt{\headlinedist} up.

\texttt{\_makefootline} appends the \texttt{\footline} to the page-body box.

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

The \texttt{\_footnoterule} is defined here.
\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.

\begin{verbatim}
\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}% increase |pageno|
\def\raggedbottom{\topskip=\dimexpr\topskip plus60pt \pgbottomskip=0pt plus1fil\relax}
\def\normalbottom{\topskip=\dimexpr\topskip\relax \pgbottomskip=0pt\relax}
\public\pageno \folio \nopagenumbers \advancepageno \raggedbottom \normalbottom ;
\end{verbatim}

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 nor \vfootnote don't take the footnote text as a parameter. This is due to a user can do catcode settings (like inline verbatim) in the footnote text. This idea is adapted from plain \TeX. The \footnote and \footstrut is defined as in plain \TeX.

\begin{verbatim}
\newinsert\footins
\def\footnote #1{% parameter \#2 (the text) is read later
  \ifhmode \edef\osf{\spacefactor\the\spacefactor}\fi
  \#1\osf\vfootnote{#1}}
\newinsert\vfootnote
\def\vfootnote{
  \opfootnote{}\insertfootins\bgroup
  \interlinepenalty=\interfootnotelinepenalty
  \leftskip=\zo \rightskip=\zo \spaceskip=\zo \xspaceskip=\zo \relax
  \resetattrs
  #1\relax % local settings used by \fnote macro
  \splittopskip=\ht\strutbox\relax % top baseline for broken footnotes
  \splitmaxdepth=\dp\strutbox\relax % maximum depth
  \textindent{#2}\footstrut
  \isnextchar\bgroup\
  \def\vfootA{\unskip\strut\egroup}
  \def\vfootB #1{#1\unskip\strut\egroup}
  \def\footstrut{\vbox to\splittopskip{}}
  \skip\footins=\bigskipamount % space added when footnote is present
  \count\footins=1000 % footnote magnification factor (1 to 1)
  \dimen\footins=8in % maximum footnotes per page
  \public\footins \vfootnote \footstrut ;
\end{verbatim}

The \topins macros \topinsert, \midinsert, \pageinsert, \endinsert are the same as in plain \TeX.
The \texttt{\draft} macro is an example of usage \texttt{\pgbackground} to create watercolor marks.

\begin{verbatim}
\def \draft {\pgbackground={\draftbox{\draftfont DRAFT}}}%
\def \draftbox #1{\setbox0=\hbox{\setgreycolor{.8}#1}%
  \kern.5\vsize \kern\voffset \kern4.5\wd0
  \hbox to0pt{\kern.5\xhsize \kern\hoffset \kern-2\wd0
    \pdfsave \pdfrotate{55} \pdfscale{10}{10}%
    \hbox to0pt{\box0\hss}%
    \pdfrestore \hss}%
}\global \let \draftfont=\draftfont
\public \draft ;
\end{verbatim}

\section{Margins}

The \texttt{\margins} macro is documented in the section \ref{sec:margins}.

\begin{verbatim}
\newdimen\pgwidth \newdimen\pgheight \pgwidth=0pt
\newdimen\shiftoffset
\def\margins/#1 #2 (#3,#4,#5,#6)#7 {
  \def\tmp{#7}
  \ifx\tmp\empty
    \opwarning{\texttt{margins: missing unit, mm inserted}}
  \else \def\tmp{mm}\fi
  \setpagedimens #2 % setting \pgwidth, \pgheight
  \ifdim\pgwidth=0pt \else
    \hoffset=0pt \voffset=0pt
    \if\texttt{#3}\if\texttt{#4}\hoffset =\dimexpr (\pgwidth -\hsize)/2 \relax
    \else \hoffset =\dimexpr \pgwidth -\hsize - #4\tmp \relax % only right margin
    \fi \fi
    \else \if\texttt{#4}\hoffset = #3\tmp \relax % only left margin
    \else \hsize =\dimexpr \pgwidth - #3\tmp - #4\tmp \relax % left+right margin
    \hoffset = #3\tmp \relax
    \else \if\texttt{#5}\hoffset = #5\tmp \relax % only top margin
    \else \vsize=\dimexpr \pgheight - #5\tmp - #6\tmp \relax % top+bottom margin
    \hoffset = #5\tmp \relax
    \else \if\texttt{#6}\voffset = #6\tmp \relax % only bottom margin
    \else \vsize=\dimexpr \pgheight - #5\tmp - #6\tmp \relax % top+bottom margin
    \voffset = #5\tmp \relax
    \fi \fi
  \fi
  \if 1\texttt{#1}\shiftoffset=0pt \else \if 2\texttt{#1} % double-page layout
    \shiftoffset = \dimexpr \pgwidth -\hsize -2\hoffset \relax
  \def\prepoffsets{\ifodd\pageno \else \advance\hoffset \shiftoffset \fi}\fi
  \else \opwarning{use \texttt{margins/1} or \texttt{margins/2}}\fi
}\fi
\def\setpagedimens{\isnextchar(\setpagedimensB}{\setpagedimensA}
\def\setpagedimensA#1 {
  \ifcsname _pgs:#1\endcsname
    \setpagedimensB \csname _pgs:#1\space\endcsname
  \else \opwarning{page specification "#1" is undefined}\fi
}\def\setpagedimensB (#1,#2)#3 {
  \setpagedimensC \pgwidth=#1:#3 \pgheight=#2:#3
  \pdfpagewidth=\pgwidth \pdfpageheight=\pgheight
}\end{verbatim}

\section{Margins}

The \texttt{\margins} macro is documented in the section \ref{sec:margins}.

\begin{verbatim}
\newdimen\pgwidth \newdimen\pgheight \pgwidth=0pt
\newdimen\shiftoffset
\def\margins/#1 #2 (#3,#4,#5,#6)#7 {
  \def\tmp{#7}
  \ifx\tmp\empty
    \opwarning{\texttt{margins: missing unit, mm inserted}}
  \else \def\tmp{mm}\fi
  \setpagedimens #2 % setting \pgwidth, \pgheight
  \ifdim\pgwidth=0pt \else
    \hoffset=0pt \voffset=0pt
    \if\texttt{#3}\if\texttt{#4}\hoffset =\dimexpr (\pgwidth -\hsize)/2 \relax
    \else \hoffset =\dimexpr \pgwidth -\hsize - #4\tmp \relax % only right margin
    \fi \fi
    \else \if\texttt{#4}\hoffset = #3\tmp \relax % only left margin
    \else \hsize =\dimexpr \pgwidth - #3\tmp - #4\tmp \relax % left+right margin
    \hoffset = #3\tmp \relax
    \else \if\texttt{#5}\hoffset = #5\tmp \relax % only top margin
    \else \vsize=\dimexpr \pgheight - #5\tmp - #6\tmp \relax % top+bottom margin
    \hoffset = #5\tmp \relax
    \else \if\texttt{#6}\voffset = #6\tmp \relax % only bottom margin
    \else \vsize=\dimexpr \pgheight - #5\tmp - #6\tmp \relax % top+bottom margin
    \voffset = #5\tmp \relax
    \fi \fi
  \fi
  \if 1\texttt{#1}\shiftoffset=0pt \else \if 2\texttt{#1} % double-page layout
    \shiftoffset = \dimexpr \pgwidth -\hsize -2\hoffset \relax
  \def\prepoffsets{\ifodd\pageno \else \advance\hoffset \shiftoffset \fi}\fi
  \else \opwarning{use \texttt{margins/1} or \texttt{margins/2}}\fi
}\fi
\def\setpagedimens{\isnextchar(\setpagedimensB}{\setpagedimensA}
\def\setpagedimensA#1 {
  \ifcsname _pgs:#1\endcsname
    \setpagedimensB \csname _pgs:#1\space\endcsname
  \else \opwarning{page specification "#1" is undefined}\fi
}\def\setpagedimensB (#1,#2)#3 {
  \setpagedimensC \pgwidth=#1:#3 \pgheight=#2:#3
  \pdfpagewidth=\pgwidth \pdfpageheight=\pgheight
}\end{verbatim}
The common page dimensions are defined here.

The \_def\_setpagedimenC \#1=\#2:\#3 \{\#1=\#2\_ifx^\#3^\_tmp\_else\#3\_fi\_relax\_truedimen\#1\}

\_public \_margins ;

The common page dimensions are defined here.

\sdef\_pgs:a3\{(297,420)mm\} \sdef\_pgs:a4\{(210,297)mm\} \sdef\_pgs:a5\{(148,210)mm\}
\sdef\_pgs:a3l\{(420,297)mm\} \sdef\_pgs:a4l\{(297,210)mm\} \sdef\_pgs:a5l\{(210,148)mm\}
\sdef\_pgs:b5\{(176,250)mm\} \sdef\_pgs:letter\{(8.5,11)in\}

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

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

\def\trueunit{} \def\magscale[#1]{\_mag=#1 \def\trueunit{true} \_ifdim\pgwidth=0pt \_else \_truedimen\pgwidth \_truedimen\pgheight \_fi \_truedimen\pdfpagewidth \_truedimen\pdfpageheight}
\def\truedimen#1{\_ifx\trueunit\empty \else\#1=\ea\ignorept\the\#1truept \_fi}

\public \magscale ;

\section{2.20 Colors}

\subsection{2.20.1 Basic concept}

Setting of color in PDF is handled by graphics operators which change the graphics context. Colors for fills/strokes are distinguished, but apart from that, only one color is active at time and is used for all material drawn by following graphics operators, until next color is set. Each PDF content (e.g. page or form XObject) has its own graphics context, that is initialized from zero. Hence we have different concept of selecting fonts in \TeX{} (it depends on \TeX{} groups but does not depends on pages) and color handling in PDF.

\TeX{} itself has no concept of colors. Colors have always been handled by inserting whatsis (either using \special for DVI or using \pdfliteral/\pdfcolorstack for PDF). It is very efficient and \TeX{} doesn't even have to know anything about colors, but it is also problematic in many ways.

That is the reason why we decided to change color handling from \pdfcolorstack to Lua\TeX{} attributes in version 1.04 of Op\TeX{}. Using attributes, the color setting behaves exactly like font selection from \TeX{} point of view: it respects \TeX{} groups, colors can span more pages, independent colors can be set for \inserts, etc. Moreover, once a material is created (using setbox for example) then it has its fonts and its colors frozen and you can rely on it when you are using e.g. \unhbox. There are no internal whatsis for colors which can interfere with other typesetting material. In the end something like setting text to red (\textcolor{red}{text}) should have the same nice behavior like setting text to bold (\textbf{text}).

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

Macros \setcmykcolor{\langle C \rangle \langle M \rangle \langle Y \rangle \langle K \rangle} or \setrgbcolor{\langle R \rangle \langle G \rangle \langle B \rangle} or \setgreycolor{\langle Grey \rangle} 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.

\subsection{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 tonners 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 {0.3\Green + 0.4\Blue \useK}
```
The \useK command exactly does:

\[
\begin{align*}
    k' &= \min(C, M, Y), \\
    C &= (C - k')/(1-k'), \\
    M &= (M - k')/(1-k'), \\
    Y &= (Y - k')/(1-k'), \\
    K &= \min(1, K + k').
\end{align*}
\]

You can use minus instead of plus in the linear combination in \colordef. The given color is substracted in such case and the negative components are rounded to zero immediately. For example

\[\colordef \Color \{ \Brown - \Black \} \]
can be used for removing the black component from the color. You can use the \-\Black trick after \useK command to remove grey components occurred during color mixing.

Finally, you can use ‘\^’ immediately preceded before the macro name of the color. Then the complementary color is used here.

\[\colordef\mycolor\{ \Grey+.6^\Blue \} \]

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:**

\[
\begin{align*}
    R &= (1-C)(1-K), \\
    G &= (1-M)(1-K), \\
    B &= (1-Y)(1-K).
\end{align*}
\]

**RGB to CMYK:**

\[
\begin{align*}
    K' &= \max(R, G, B), \\
    C &= (K' - R)/K', \\
    M &= (K' - G)/K', \\
    Y &= (K' - B)/K', \\
    K &= 1 - K'.
\end{align*}
\]

The RGB to CMYK conversion is invoked when a color is declared using \setrgbcolor and it is used in \colordef or if it is printed when \onlycmyk is declared. The CMYK to RGB conversion is invoked when a color is declared using \setcmykcolor and it is used in \rgbcolordef or if it is printed when \onlyrgb is declared.

### 2.20.3 Implementation

By default, the \setcmykcolor \setrgbcolor and \setgreycolor macros with \{⟨componets⟩\} parameter expand to \setcolor{⟨color-data⟩}{⟨fill-op⟩}{⟨stroke-op⟩} where ⟨color-data⟩ is {⟨R⟩ ⟨G⟩ ⟨B⟩ or ⟨C⟩ ⟨M⟩ ⟨Y⟩ ⟨K⟩} and ⟨fill-op⟩ and ⟨stroke-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 \rbgcolordef.

\begin{verbatim}
47 \def\onlyrgb{\_def\Red{\_setrgbcolor{1 0 0}}}%
48 \_def\Green{\_setrgbcolor{0 1 0}}% \_def\Blue{\_setrgbcolor{0 0 1}}%
49 \_let\_colordef=\_rgbcolordef
50 \_def\_setrgbcolor##1{\_setcolor{##1}{rg}{RG}}%
51 \_def\_setcmykcolor##1{\_ea\_setcolor\_ea{\_expanded{\_cmyktorgb ##1 ;}}kK}%
52 \_public \colordef \setrgbcolor \setcmykcolor ;}
53 \_def\_onlycmyk{\_let\_colordef=\_cmykcolordef
54 \_def\_setrgbcolor##1{\_ea\_setcolor\_ea{\_expanded{\_rgbtocmyk ##1 ;}}kK}%
55 \_def\_setcmykcolor##1{\_setcolor{##1}kK}%
56 \_public \colordef \setrgbcolor \setcmykcolor ;}
57 \_public \onlyrgb \onlycmyk ;
\end{verbatim}

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:

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

\begin{verbatim}
77 \_newattribute \_colorattr
78 \_newcount \_colorcnt \_colorcnt=1 % allocations start at 1
79 \_protected\_def\_setcolor{\_colorprefix\_colorattr=\_translatecolor}
80 \_def\_translatecolor#1#2#3{\_ifcsname _color::#1 #2\_endcsname\_lastnamedcs\_relax
81 \_else
82 \_colorcnt
83 \_sxdef{\_color::#1 #2}{\_the\_colorcnt}%
84 \_sxdef{\_color:#1 #2}{\_the\_colorcnt}\_noexpand\_noattr
85 \_sxdef{\_color-s:#1 #3}{\_the\_colorcnt}\_noattr
86 \_incr \_colorcnt
87 \_fi
88 }
89 % Black is the default color.
90 \_sdef{\_color::0 g}{0}
91 \_sdef{\_color:0}{0 g}
92 \_sdef{\_color-s:0}{0 G}
\end{verbatim}

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.

\begin{verbatim}
102 \_protected\_def \_localcolor {\_let\_colorprefix=\_relax}
103 \_protected\_def \_nolocalcolor {\_let\_colorprefix=\_global}
104 \_public \localcolor \nolocalcolor ;
105 \_localcolor
\end{verbatim}

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 \(255-(\text{number})/255\)). This information is used when shipout is processed (similarly as colors). It means /tr\{number\} gs is inserted when the attribute is changed.

\_resetattrs resets the \_colorattr and \_transpattr to their initial value "7FFFFFFF.

\begin{verbatim}
119 \_newattribute \_transpattr
120 \_def \_transparency {\_afterassignment\_transparencyA \_transpattr}
121 \_def \_transparencyA{%
122 \_ifnum\_transpattr<1 \_transpattr=\_noattr \_fi
123 \_ifnum\_transpattr>255 \_opwarning{\_noexpand\transparency > 255 not allowed}%
124 \_transpattr=\_noattr
125 \_else
126 \_ifcsname \_transpattr\_the\_transpattr\_endcsname \_else
127 \_edef \_transpv{\_expr{(255-\_the\_transpattr)/255)}%}
\end{verbatim}

111
We use Lua codes for RGB to CMYK or CMYK to RGB conversions and for addition color components in the \colordef macro. The \_rgbtocmyk \( \langle R \rangle \langle G \rangle \langle B \rangle \) expands to \( \langle C \rangle \langle M \rangle \langle Y \rangle \langle K \rangle \) and the \_cmyktorgb \( \langle C \rangle \langle M \rangle \langle Y \rangle \langle K \rangle \) expands to \( \langle R \rangle \langle G \rangle \langle B \rangle \). 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: or \torgb: 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.

\def\_rgbtocmyk #1 #2 #3 ;\{\_trycs{\tocmyk:#1 #2 #3}{}\}
\def\_cmyktorgb #1 #2 #3 #4 ;\{\_trycs{\torgb:#1 #2 #3 #4}{}\}
\def\_colorcrop\{\directlua{local m=math.max(color_C, color_M, color_Y, color_K)
if (m>1) then
    color_C=color_C/m color_M=color_M/m color_Y=color_Y/m color_K=color_K/m
end
}\}
\def\_colordefFin\{\_colorcrop \_ea \_stripzeros \_ea\{\directlua{print(string.format('pcent.3f pcent.3f pcent.3f pcent.3f ;
    (kr-#1)/kr, (kr-#2)/kr, (kr-#3)/kr, 1-kr))\end\}
\}
\def\_douseK\{\_colorcrop \directlua{local kr=math.min(color_C, color_M, color_Y)
if (kr>1) then
    color_C=0 color_M=0 color_Y=0 color_K=1
else
    color_C=(color_C-kr)/(1-kr) color_M=(color_M-kr)/(1-kr)
color_Y=(color_Y-kr)/(1-kr) color_K=math.min(color_K+kr,1)
end\}}

\def\_stripzeros #1.#2 #3{\ifx0#1\else#1\fi.\_stripzeros #2 0 :\%
\ifx;#3\else \_space \_ea\_stripzeros\_ea#3\fi}
\def\_stripzeros #10 #2{\ifx^#2^\_stripzeros #1:\else #1\fi}
\def\_stripzeros #10 #2{\ifx#2\_stripzeros #1:\else #1\fi}
\def\_stripzeros #1#2{(#1)}
\rgbcolordef \{\{R\} \{G\} \{B\}\}\{\{C\} \{M\} \{Y\} \{K\}\}\} declares mapping from RGB to CMYK and from CMYK to RGB for given color. It has precedence before general formulae used in the \rgbtocmyk and \cmyktorgb macros. Note, that the values \{R\} \{G\} \{B\} \{C\} \{M\} \{Y\} \{K\} must be given exactly in the same format as in \setcmykcolor and \setrgbcolor parameters. For example, 0.5 or .5 or .50 are different values from point of view of this mapping.

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

\input data
expanded \typetmp to:
\input data
adjusted to: \input data
\input data

and this is processed.

\\_
\_addcolor \{coef\}\!\{\mod\}\{\type\} expands to \_addcolor:\{\mod\}\{\type\} \{\coef\} for example it expands to \_addcolor:\{K\} \{\coef\} followed by one or three or four numbers (depending on \{\type\}). \{\mod\} is = (use as is) or ^ (use complementary color). \{\type\} is K for CMYK, R for RGB and G for GREY color space. Uppercase \{\type\} informs that \cmykcolordef is processed and lowercase \{\type\} informs that \rgbcolordef is processed. All variants of commands \_addcolor:\{\mod\}\{\type\} are defined. All of them expand to \addcolor\{\\{v1\} \{\v2\} \{\v3\} \{\v4\}\} which adds the values of Lua variables. The \rgbcolordef uses \addcolor\{\{R\}\} \{\{G\}\} \{\{B\}\} 0 and \cmykcolordef uses \addcolor\{\{C\}\} \{\{M\}\} \{\{Y\}\} \{\{K\}\}. So the Lua variable names are a little confusing when \rgbcolordef is processed.

Next, \_commoncolordef saves resulting values from Lua to \_tmpb using \_colordefFin. If \rgbcolordef is processed, then we must to remove the last \{K\} component which is in the format .0 in such case. The \_stripK macro does it. Finally, the \{what\-define\} is defined as \{\selector\}\{\{expanded \_tmpb\}\}, for example \setcmykcolor{1 0 .5 .3}.

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

\input data
expanded \typetmp to:
\input data
adjusted to: \input data
\input data

and this is processed.

\_addcolor \{coef\}\!\{\mod\}\{\type\} expands to \_addcolor:\{\mod\}\{\type\} \{\coef\} for example it expands to \_addcolor:\{K\} \{\coef\} followed by one or three or four numbers (depending on \{\type\}). \{\mod\} is = (use as is) or ^ (use complementary color). \{\type\} is K for CMYK, R for RGB and G for GREY color space. Uppercase \{\type\} informs that \cmykcolordef is processed and lowercase \{\type\} informs that \rgbcolordef is processed. All variants of commands \_addcolor:\{\mod\}\{\type\} are defined. All of them expand to \addcolor\{\\{v1\} \{\v2\} \{\v3\} \{\v4\}\} which adds the values of Lua variables. The \rgbcolordef uses \addcolor\{\{R\}\} \{\{G\}\} \{\{B\}\} 0 and \cmykcolordef uses \addcolor\{\{C\}\} \{\{M\}\} \{\{Y\}\} \{\{K\}\}. So the Lua variable names are a little confusing when \rgbcolordef is processed.

Next, \_commoncolordef saves resulting values from Lua to \_tmpb using \_colordefFin. If \rgbcolordef is processed, then we must to remove the last \{K\} component which is in the format .0 in such case. The \_stripK macro does it. Finally, the \{what\-define\} is defined as \{\selector\}\{\{expanded \_tmpb\}\}, for example \setcmykcolor{1 0 .5 .3}.
Public versions of \colordef and \useK macros are declared using _def, because the internal versions \colordef and \useK are changed during processing.

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

\def\vr{|\hrule height10pt depth2pt width20pt}  
\def\showcolor{|\hbox{tt\hrulefill} \vr\space \space}  
\begmulti 4 \typsize[10/14] \morecolors \endmulti

2.21 The .ref file

A so called .ref (\jobname.ref) file is used to store data that will be needed in the next \TeX run (information about references, TOC lines, etc.). If it exists it is read by \everyjob, when processing of the document starts, but it is not created at all if the document doesn’t need any forward references. Here are the typical contents of a .ref file:

\Xrefversion{⟨ref-version⟩}  
\Xpage{⟨pageno⟩}  
\Xtoc{⟨level⟩}{⟨type⟩}{⟨text⟩}{⟨title⟩}  
\Xlabel{⟨label⟩}{⟨text⟩}  
\Xlabel{⟨label⟩}{⟨text⟩}  
\Xpage{⟨pageno⟩}  
\Xlabel{⟨label⟩}{⟨text⟩}  

...
• \_xpage corresponds to the beginning of a page. \langle pageno \rangle is an internal page number, globally numbered from one. \langle the\pageno \rangle is the page number \langle \the\pageno \rangle used in pagination (they may differ).

• \_xtoc corresponds to a chapter, section or subsection title on a page. \langle title \rangle is the title of the chapter (\langle level\rangle = 1, \langle type\rangle = \text{chap}), section (\langle level\rangle = 2, \langle type\rangle = \text{sec}) or subsection (\langle level\rangle = 3, \langle type\rangle = \text{secc}).

• \_xlabel corresponds to a labelled object on a page. \langle label \rangle is the label provided by the user in \label{\langle label \rangle}, while \langle text \rangle 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}\_ifalse
  \input{\jobname.ref}\%\n  \edef{\_prevrefhash{\mdfive{\jobname.ref}}}\%
  \gfnotenum=0 \_lfnotenum=0 \_mnotenum=0
  \_openref
  \_fi
}
\_mdfive{\langle file \rangle} 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{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.

\_wrefrelax#1#2{}\_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 {\langle macro\rangle\{\langle data\rangle\}} so that it saves the line \langle \text{macro}\rangle\langle \text{data}\rangle to the .ref file using the asynchronous \write primitive. Finally, \_openref destroys itself, because we don’t need to open the file again.

\_wref\langle\text{csname}\rangle\{\langle\text{params}\rangle\} in fact does \write\reffile{\langle\text{csname}\rangle\langle\text{params}\rangle} and similarly \_ewref\langle\text{csname}\rangle\{\langle\text{params}\rangle\} does \write\reffile{\langle\text{csname}\rangle\langle\text{expanded-params}\rangle}.

\_def\_openref {\
  \_immediate\openout\reffile="\jobname.ref"\_relax
  \_edef\_reffile{\_bslash\csstring\jobname.ref}\%
  \_immediate\write\reffile{\_bslash\csstring\_REFversion}\%
  \_immediate\wref\_Xrefversion{\langle\text{REFversion}\rangle}\%
  \_immediate\wref\_empty \_else \_reffcontent \_fi
  \_gdef\_openref{}\%
}

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

\_Xrefversion{\langle\text{version}\rangle}
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.

\begin{verbatim}
\def\REFversion{6} % current version of .ref files in OpTeX
\def\Xrefversion#1{\ifnum #1=\REFversion \relax \else \endinput \fi}
\public \Xrefversion ; % we want to ignore .ref files generated by OPmac
\end{verbatim}

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:

\begin{verbatim}
\refdecl{%
def\totlist{} \def\tofllist{} ^^J
  \def\Xtab#1#2#3{\addto\totlist{\totline{#1}{#2}{#3}}} ^^J
  \def\Xfig#1#2#3{\addto\tofllist{\tofline{#1}{#2}{#3}}}}
\end{verbatim}

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.

\begin{verbatim}
\def\refdecldata{}
\def\refdecl{\bgroup \catcode`#=12 \catcode`\=12 \catcode`=12 \refdeclA}
\def\refdeclA#1{\egroup
  \ifx\refdecldata\empty
    \else \global\addto\refdecldata{^^J}\fi
  \global\addto\refdecldata{#1}%
  \ifx\openref\empty \refdeclwrite \fi
}
\def\refdeclwrite{\immediate\write\reffile{\_pcent\space \string\refdecl:^^J\detokenize\eaf{\_refdecldata}}%
  \gdef\refdecldata{}}
\public \refdecl ;
\end{verbatim}

2.22 References

If the references are “forward” (i. e. the \ref is used first, the destination is created later) or if the reference text is page number then we must read \.ref file first in order to get appropriate information. See section 2.21 for more information about \.ref file concept.

\begin{verbatim}
\codedecl \ref {References <2021-04-13>} % preloaded in format
\Xpage {\(gpageno\)}{\(pageno\)} saves the parameter pair into \_currpage. Resets \_lfnotenum; it is used if footnotes are numbered from one at each page.
\end{verbatim}

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:

\begin{verbatim}
\addto\byehook{\_ifnum\unresolvedrefs>0 \opwarning
  \{There are \_the\_unresolvedrefs\_space unresolved references\}\_fi}
\end{verbatim}

\begin{verbatim}
\newcount\unresolvedrefs
\unresolvedrefs=0
\Xlabel {\{label\}}{\{text\}} saves the \{text\} to \_lab:\{label\} and saves \{(\_pageno)\,(\_pageno)\} to \_pgref:\{label\}.
\end{verbatim}
\label{\lab} saves the declared label to \_lastlabel and \wlabel{\text} uses the \_lastlabel and activates \wref\_Xlabel{\lab}{\text}.

\ref{\lab}{\text} prints (linked) \text. The missing optional \{\text\} is replaced by \@. The \@ is replaced by \langle implicit-text \rangle from saved \lab: \ref{\lab} using \reftext macro. If the reference is backward then we know \lab: \ref{\lab} without any need to read REF file. On the other hand, if the reference is forwarded, then we don’t know \lab: \ref{\lab} in the first run of \TeX{} and we print a warning and do \openref.

\pgref{\lab}{\text} prints \text where \@ is replaced by \langle pageno \rangle. Data in the format \{\langle pageno \rangle\}\{\langle pageno \rangle\} are read from \pgref\ref{\lab} by \pgrefB{\langle pageno \rangle}{\langle pageno \rangle}{\text} and then you need not repeat the same label in typical situations and you can write for instance: \text see section \ref{lab} at page \pg.

\reftext{\langle implicit-text \rangle}{\text} expands to the \text but the optional \@ in the \text is replaced by the \reftext first.

\showlabels redefines it as box with zero dimensions and with left lapped \lab in blue 10pt \tt font shifted up by 1.7ex.
2.23 Hyperlinks

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

- \texttt{\ref}:〈label〉 – the destination is created when \texttt{\label}[(label)] is used, see also the section 2.22.
- \texttt{\toc}:[tocrefnum] – the destination is created at chap/sec/secc titles, see also the section 2.24.
- \texttt{\pg}:[gpageno] – the destination is created at beginning of each page, see also the section 2.18.
- \texttt{\cite}:[bibnum] – the destination is created in bibliography reference, see section 2.32.1.
- \texttt{\fn}:[gfnotenum] – link from footnote to text, see also section 2.34.
- \texttt{\url}:[url] – used by \url or \ulink, see also the end of this section.

The \texttt{\tocrefnum}, \texttt{\gpageno}, \texttt{\bibnum}, and \texttt{\gfnotenum} are numbers starting from one and globally incremented by one in the whole document. The registers \texttt{\tocrefnum}, \texttt{\gpageno}, \texttt{\bibnum}, and \texttt{\gfnotenum} are used for these numbers.

When a chap/sec/secc title is prefixed by \texttt{\label}[(label)], then both types of internal links are created at the same destination place: \texttt{\toc}:[tocrefnum] and \texttt{\ref}:[label].

The color for active links can be declared by \texttt{\def\_linkcolor{⟨color⟩}}. These macros are not declared by default, so color for active links are given only by \texttt{\textbf{\hyperlinks}} macro and borders are invisible. For example \texttt{\def\toclinkcolor{\textbf{Red}}} means that links from table of contents are in red. Another example \texttt{\def\tocborder{0 0 0}} causes red frames in TOC (not printed, only visible in PDF viewers).

\begin{Verbatim}
\def\tocborder{0 0 0} \def\toclinkcolor{\textbf{Red}}
\end{Verbatim}

Each hyperlink is created internally by \texttt{\_linkdimens}. This macro expands to \texttt{\quitvmode}⟨text⟩ by default, i.e. no active hyperlink is created, only ⟨text⟩ is printed in horizontal mode (and in a group). If \texttt{\hyperlinks} is used, then \texttt{\_linkdimens} gets the meaning of \texttt{\linkactive} and hyperlinks are created by the \texttt{\pdfstartlink} and \texttt{\pdfendlink} primitives. The ⟨text⟩ has given ⟨color⟩ only when hyperlink is created. If \texttt{\_linkcolor} is defined, it has precedence over ⟨color⟩.

The \texttt{\linkdimens} macro declares the dimensions of link area.

A specific action can be defined for each link ⟨type⟩ by the macro \texttt{\_urlaction{⟨url⟩}}. Op\TeX defines only \texttt{\urlaction{⟨url⟩}}. The default link action (when \texttt{\_urlaction} is not defined) is \texttt{\nofollow}. Op\TeX is implemented by \texttt{\pdfstartlink}⟨type⟩⟨spec⟩⟨color⟩⟨text⟩. This macro expands to \texttt{\pdfstartlink}⟨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 \texttt{\hyperlinks} is used, then \texttt{\_urlaction} gets the meaning of \texttt{\linkactive} and hyperlinks are created by the \texttt{\pdfstartlink} and \texttt{\pdfendlink} primitives. The ⟨text⟩ has given ⟨color⟩ only when hyperlink is created. If \texttt{\_linkcolor} is defined, it has precedence over ⟨color⟩.

The \texttt{\linkdimens} macro declares the dimensions of link area.

A specific action can be defined for each link ⟨type⟩ by the macro \texttt{\_urlaction{⟨url⟩}}. Op\TeX defines only \texttt{\urlaction{⟨url⟩}}. The default link action (when \texttt{\_urlaction} is not defined) is \texttt{\nofollow}. Op\TeX is implemented by \texttt{\pdfstartlink}⟨type⟩⟨spec⟩⟨color⟩⟨text⟩. This macro expands to \texttt{\pdfstartlink}⟨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 \texttt{\hyperlinks} is used, then \texttt{\_urlaction} gets the meaning of \texttt{\linkactive} and hyperlinks are created by the \texttt{\pdfstartlink} and \texttt{\pdfendlink} primitives. The ⟨text⟩ has given ⟨color⟩ only when hyperlink is created. If \texttt{\_linkcolor} is defined, it has precedence over ⟨color⟩.
\texttt{\textbackslash link\{\textbackslash type\}:\{\textbackslash spec\}\{\textbackslash color\}\{\textbackslash text\}} creates a link. It is kept here for backward compatibility and it is equivalent to \texttt{\_xlink\{\textbackslash type\}\{\textbackslash spec\}\{\textbackslash color\}\{\textbackslash text\}}. If \texttt{\textbackslash type\{\textbackslash action\}} is not defined then \texttt{\link} creates internal link do the \texttt{\dest\{\textbackslash type\}:\{\textbackslash spec\}}. You can have more links with the same \texttt{\textbackslash type\:\{\textbackslash spec\}} but only one \texttt{\dest} in the document.

\texttt{\textbackslash linke\{\textbackslash type\}:\{\textbackslash spec\}\{\textbackslash text\}} is equivalent to \texttt{\link} but the \texttt{\dest} is used from \texttt{\hyperlinks} declaration (or it is overwritten by \texttt{\\textbackslash def\{\textbackslash type\\textbackslash linkcolor\}}).

\texttt{\url\{\textbackslash ur\}}\{\textbackslash text\}\{\textbackslash url\}}\{\textbackslash url\}} creates external link. The \texttt{\url\}} is detokenized with \texttt{\escapechar=-1} before it is used, so \%\%, \# etc. can be used in the \texttt{\url\}}.

\texttt{\hyperlinks\{\textbackslash link color\}\{\textbackslash ulink color\}} activates \texttt{\dest}, \texttt{\xlink}, so that they create links. Not setting colors \texttt{\hyperlinks} is also supported.

\texttt{\url\}} does approximately the same as \texttt{\url\}}\{\textbackslash url\}}\{\textbackslash url\}}\{\textbackslash url\}} but more work is done before the \texttt{\url\}} is processed. The link-version of \texttt{\url\}} is saved to \texttt{\_tmpa} and the printed version in \texttt{\_tmpb}. The printed version is processed in four steps: 1. the \texttt{\_} are replaced by \texttt{[11]} (we suppose that such string does not exist in any URL). 2. it is detokenized with \texttt{\escapechar=-1}. 3. \texttt{\_urlA}, \texttt{\_urlB} and \texttt{\_urlC}. The step 4 do following: The \texttt{\_urlskip} is inserted between each pair of “normal characters”, i.e. characters not declared by \texttt{\_urlC}. The special characters declared by \texttt{\_def\_ur\{\textbackslash character\}} are replaced by the body of their corresponding macro. The \texttt{\_urlskip}, \texttt{\_urlbkip}, \texttt{\_urlgskip} are typical skips used for special characters, their meaning is documented in the code below. You can change them. Default values: penalty 9990 is inserted between each pair of normal characters, penalty 100 is inserted after special characters, nobreak before special characters. The URL can be broken at any place using these default values. If you want to disable breaking between normal characters, say \texttt{\_let\url\_urlskip\_nobreak}. The text version of the \texttt{\url\}} is printed in \texttt{\_urlfont}.

---

[119]
2.24 Making table of contents

\maketoc {Macros for maketoc <2021-07-18>} % preloaded in format

\_Xtoc \{\langle level\rangle\} \{\langle type\rangle\} \{\langle number\rangle\} \{\langle o-title\rangle\} \{\langle title\rangle\} \{\langle gpageno\rangle\} \{\langle pageno\rangle\}

where:

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

The last two parameters are restored from previous \_Xpage\{\langle pageno\rangle\} \{\langle gpageno\rangle\}, data were saved in the \_currpage macro.

We read the \langle title\rangle parameter by \scantoeol from .ref file because the \langle title\rangle can include something like `{`.

\_tocline \{\langle level\rangle\} \{\langle type\rangle\} \{\langle number\rangle\} \{\langle o-title\rangle\} \{\langle title\rangle\} \{\langle gpageno\rangle\} \{\langle pageno\rangle\}

prints the record to the table of contents. It opens group, reduces \_leftskip, \_rightskip, runs the \everytocline (user can customise the design of TOC here) and runs \_tocl: \{\langle level\rangle\} \{\langle number\rangle\} \{\langle title\rangle\} \{\langle gpageno\rangle\} \{\langle pageno\rangle\} macro. This macro starts with vertical mode, inserts one record with given \langle level\rangle and it should end by \_tocpar which returns to horizontal mode. The \_tocpar appends \_nobreak \_hskip-2\_indent\_null \_par. This causes that the last line of the record is shifted outside the margin given by \_rightskip. A typical record (with long \langle title\rangle) looks like this:

| | text text text text text
| text text text text text
text text text text text text | \langle pageno\rangle

Margins given by \_leftskip and \_rightskip are denoted by | in the example above. \tocrefnum is the global counter of all TOC records (used by hyperlinks).
You can re-define default macros for each level of tocline if you want. Parameters are \{(number)\}\{(title)\}\{(pageno)\}.

The auxiliary macros are:

- \_llaptoclink\{(text)\} does \_noindent\_llap\{(linked text)\}.
- \_tocdotfill creates dots in the TOC.
- \_nofirst\macro applies the \macro only if we don’t print the first record of the TOC.
- \_tocpar finalizes one TOC record with \hfill\_pgn\{(pageno)\}.
- \_pgn\{(pageno)\} creates \(\text{<pageno>}\) as link to real \gpage saved in \_tocrefnum of \_tocline. This is temporarily defined in the \_tocline.

If you want a special formatting of TOC with adding more special lines (no generated as titles from \chap, \sec, \secc), you can define \addtotoc\{\{level\}\}\{\{type\}\}\{(number)\}\{(o-title)\}\{(title)\} macro:

\def\addtotoc#1#2#3#4#5{\ incr\_tocrefnum
 \_dest[\{toc:\_the\_tocrefnum\}]{\_enspace#1\_kern.4em}\_kern.1em]
 \_ewref\_Xtoc\{#1\{#2\#3\#4\#5\}
}

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

\addtotoc\{10\}{blue-line}{}{}{blue text to be added in the TOC}

Now, users can add a blue line into TOC by

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

maketoc.opm

\newtoks \_regtoc \newtoks \_regmark \newtoks \_regoul
\def \_regmacro #1#2#3{% 
  \_toksapp \_regtoc{#1} \_toksapp \_regmark{#2} \_toksapp \_regoul{#3}%
}\public \maketoc \_regmacro ;

2.25 PDF outlines

2.25.1 Nesting PDF outlines

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

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

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

\def \_outlinelevel{#1}\_let \_tocline=\_outlinesB
\def \_count0=0 \_count1=0 \_count2=0 \_count3=0
\_toclist % create outlines

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.

outlines.opm

\_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}%
\_else
\_ifx \_dest \_destactive \_else
\_opwarning{\_noexpand \_outlines doesn't work when \_noexpand \_hyperlinks isn't declared}\_fi
\{\_let \_tocline=\_outlinesA
\_count0=0 \_count1=0 \_count2=0 \_count3=0 \_toclist % calculate numbers o childs
\_def \_outlinelevel{#1}\_let \_tocline=\_outlinesB
\_tocrefnum=0 \_count0=0 \_count1=0 \_count2=0 \_count3=0
\_toclist % create outlines
\_fi
\}
\_def \_outlines#1\{\_pdfcatalog{/PageMode/UseOutlines}\_openref
\_isequal{\_relax}{#1}\_iffalse
\_advance \_count#1 by1
\_ifcase#1 \_or
\}
\_def \_outlines#1\{\_pdfcatalog{/PageMode/UseOutlines}\_openref
\_isequal{\_relax}{#1}\_iffalse
\_advance \_count#1 by1
\_ifcase#1 \_or

2.25.2 Strings in PDF outlines

There are only two encodings for PDF strings (used in PDFoutlines, PDFinfo, etc.). The first one is PDFDocEncoding which is single-byte encoding, but it misses most international characters.

The second encoding is Big Endian UTF-16 which is implemented in this file. It encodes a single character in either two or four bytes. This encoding is TEX-discomfortable because it looks like

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

This example shows a hexadecimal PDF string (enclosed in <> as opposed to the literal PDF string enclosed in ()). In these strings each byte is represented by two hexadecimal characters (0-9, A-F). You can tell the encoding is UTF-16BE, because it starts with “Byte order mark” FFEB. Each unicode character is then encoded in one or two byte pairs. The example string corresponds to the text “Cvičení je zátěž a $x \in \mathbb{M}$”. Notice the 4 bytes for the last character, $\mathbb{M}$. (Even the whitespace would be OK in a PDF file, because it should be ignored by PDF viewers, but LuaTEX doesn’t allow it.)

This 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.
local high = bit32.rshift(num, 10) + 0xD800
local low = bit32.band(num, 0x3FF) + 0xDC00
tex.print(string.format("%04X%04X", high, low))
}
}
}

\pdfunidef\macro{\langle text \rangle} defines \macro as \langle text \rangle converted to Big Endian UTF-16 and enclosed to <>. Example of usage: \pdfunidef\infoauthor{Petr Olšák} \pdfinfo{/Author \infoauthor}.
\pdfunidef does more things than only converting to hexadecimal PDF string. The \langle text \rangle can be scanned in verbatim mode (it is true because \_Xtoc reads the \langle text \rangle in verbatim mode). First \edef do \_scantextokens\unexpanded and second \edef expands the parameter according to current values on selected macros from \_regoul. Then \_removeoutmath converts ..$x^2$.. to ..x^2.., i.e removes dollars. Then \_removeoutbraces converts ..(x) to ..x.. Finally, the \langle text \rangle is detokenized, spaces are preprocessed using \_replstring and then the \_pdfunidefB is repeated on each character. It calls the \directlua chunk to print hexadecimal numbers in the macro \_hexprint.

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.


2.2.6 Chapters, sections, subsections

We are using scaled fonts for titles \titfont, \chapfont, \secfont and \seccfont. They are scaled from main fonts size of the document, which is declared by first \typosize[⟨fo-size]/⟨b-size⟩] command.

The \tit macro is defined using \scantoeol and \titprint. 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{(penaltyA){⟨skipA⟩}} and \belowtitle{⟨skipB⟩} for inserting vertical material above and below the section title. The arguments of these macros are normally used, i.e. \abovetitle inserts ⟨penaltyA⟩⟨skipA⟩ and \belowtitle inserts ⟨skipB⟩. But there is an exception: if \belowtitle{⟨skipB⟩} is immediately followed by \abovetitle{(penaltyA){⟨skipA⟩}} (for example section title is immediately followed by subsection title), then only ⟨skipB⟩ is generated, i.e. ⟨skipB⟩⟨penaltyA⟩⟨skipA⟩ is reduced only to ⟨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 ⟨belowtitle⟩) and generates only greater from this pair of skips. It means that ⟨whatever-skip⟩⟨penaltyA⟩⟨skipA⟩ is transformed to ⟨penaltyA⟩max(⟨whatever-skip⟩⟨skipA⟩). The reason for such behavior: we don’t want to duplicate vertical skips (from \belowlistskip, for example) above the title.

- Use \printrefnum[⟨pre⟩@⟨post⟩] in horizontal mode. It prints ⟨pre⟩⟨ref-num⟩⟨post⟩. The ⟨ref-num⟩ is \thechapnum or \thesecnum or \theseccnum depending on what type o 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⟩] preceeds) and saves references for the table of contents (if \maketoc is used).

- Use \nbpar for closing the paragraph for printing title. This command inserts \nobreak between each line of such paragraph, so the title cannot be broken into more pages.

- You can use \firstnoindent in order to the first paragraph after the title is not indented.

\let\space= \public \pdfunidef ;

\public \pdfunidef ;

\def \titfont { \scalemain[typoscale[⟨magstep4⟩/⟨magstep5⟩]\boldify}
 \def \chapfont { \scalemain[typoscale[⟨magstep3⟩/⟨magstep3⟩]\boldify}
 \def \secfont { \scalemain[typoscale[⟨magstep2⟩/⟨magstep2⟩]\boldify}
 \def \seccfont { \scalemain[typoscale[⟨magstep1⟩/⟨magstep1⟩]\boldify}

\def \printtit #1{\vglue\titskip
\leftskip=0pt plus1fill \rightskip=\leftskip % centering
\titfont \noindent \scantextokens{#1}\par}

\def \tit{\scantoeol \printtit}

\def \printchap #1{\vglue\medskipamount % shifted by topkip+\medskipamount
\chapfont \noindent \mtext{chap} \printrefnum[@]\par
\nobreak \bigskip}

\def \printsec #1{\secfont \noindent \raggedright #1\nbpar}\mark{}

\def \printsecc #1{\seccfont \noindent \raggedright #1\nbpar}\mark{\bigskip}

\def \firstnoindent
\public \tit ;

\public \tit ;
The \_sectionlevel is the level of the printed section:

- \_sectionlevel=0 – reserved for parts of the book (unused by default)
- \_sectionlevel=1 – chapters (used in \chap)
- \_sectionlevel=2 – sections (used in \sec)
- \_sectionlevel=3 – subsections (used in \secc)
- \_sectionlevel=4 – subsubsections (unused by default, see the OpTeX trick 0033)

The \_chapx initializes counters used in chapters, the \_secx initializes counters in sections and \_seccx initializes counters in subsections. If you have more types of numbered objects in your document then you can declare appropriate counters and do \addto\_chapx{\yourcounter=0} for example. If you have another concept of numbering objects used in your document, you can redefine these macros. All settings here are global because it is used by \{\_globaldefs=1 \_chapx\}.

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

The \_the* macros \_thechapnum, \_thesechnum, \_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 \langle num\rangle and subsections \langle num\rangle. On the other hand, if chapter is used in the document then \_chapnum>0 and sections have numbers \langle num\rangle. \langle num\rangle and subsections have numbers \langle num\rangle. \langle num\rangle. \langle num\rangle.

The \notoc and \nonum prefixes are implemented by internal \_ifnotoc and \_ifnonum. They are reset after each chapter/section/subsection by the \_resetnonumnotoc macro.
The \chap, \sec, and \secc macros are implemented here. The \inchap, \insec and \insecc macros do the real work. First, we read the optional parameter \[\langle label\rangle\], if it exists. The \chap, \sec and \secc macro reads its parameter using \texttt{scantoeol}. This causes that they cannot be used inside other macros. Use \texttt{\inchap}, \texttt{\insec}, and \texttt{\insecc} macros directly in such case.

\begin{verbatim}
\setcounter{foo}{0}
\chap{The \texttt{\inchap} macro is used in \texttt{\inchap} macros. Translate before \_wref, so the problem of “fragile macros” from old \LaTeX{} never occurs. This fourth parameter is not delimited by \{...\} but by end of line. This gives possibility to have unbalanced braces in inline verbatim in titles.
\end{verbatim}

The \texttt{\printrefnum[\langle pre\rangle\#\langle post\rangle]} macro is used in \texttt{\print*} macros.

Note that the \texttt{\langle tite-text\rangle} is \texttt{detokenize}d before \texttt{\_wref}, so the problem of “fragile macros” from old \LaTeX{} never occurs. This fourth parameter is not delimited by \dots{} but by end of line. This gives possibility to have unbalanced braces in inline verbatim in titles.

The \texttt{\_abovetitle{\langle penaltyA\rangle}-\texttt{\langle skipA\rangle}} and \texttt{\belowtitle{\langle skipB\rangle}} pair communicates using a special penalty 11333 in vertical mode. The \belowtitle puts the vertical skip (its value is saved in \texttt{\_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 \texttt{\_save\_last\_skip} only if whatever previous skip is less or equal than \texttt{\langle skipA\rangle}. We must save \texttt{\_whatever\_skip}, remove it, create \texttt{\_last\_skip} (if \belowtitle does not precede) and create \texttt{\_whatever\_skip} or \texttt{\_last\_skip} depending on what is greater. The amount of \texttt{\langle skipA\rangle} is measured using \texttt{\setbox0=\vbox{}}.
\_nbpar sets \_interlinepenalty value. \_nl is “new line” in the text (or titles), but space in toc or headlines or outlines.

\_firstnoindent puts a material to \_everypar in order to next paragraph will be without indentation. It is useful after titles. If you dislike this feature then you can say \_let \_firstnoindent=\_relax. The \_wipeepar removes the material from \_everypar.

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 \_marks. And they can be mixed with chapter’s running heads, of course.

The \_insertmark{\langle title text \rangle} saves \_mark in the format {\langle title-num \rangle} {\langle title-text \rangle}, so it can be printed “as is” in \_headline (see the space between them), or you can define a formatting macro with two parameters for processing these data, if you need it.

OpTeX sets \_headline={} by default, so no running headings are printed. You can activate the running headings by following code, for example. See also issue 100.

\_caption{\langle letter \rangle} increases \_\langle letter \rangle num counter, edefines \_thecapnum as \_the \_\langle letter \rangle num and defines \_thecaptitle as language-dependent word using \_mtext, runs the \_everycaption{\langle letter \rangle} tokens register. The group opened by \_caption is finalized by first \_\par from an empty line or from
or from \endinsert. The \_printcaption(letter) is called, it starts with printing of the caption. The \_skip macro inserts nonbreakable vertical space between the caption and the object.

\begin{verbatim}
\_def \_caption/#1{\_def \_tmpa{#1} \_nospaceafter \_capA}
\_optdef \_capA []{}{\_trylabel \_incipation}
\_def \_incipation {}{\_bgroup}
\_ifcsname \_\tmpa num\_endcsname \_ea \_incr \_csname \_\tmpa num\_endcsname
\_else \_opwarning{Unknown caption /\_\tmpa}\_fi
\_edef \_thecapnum {\_csname \_the\_tmpa num\_endcsname}\%
\_edef \_thecaptitle {\_mtext{\_tmpa}}\%
\_ea \_the \_csname \_everycaption\_\tmpa\_endcsname
\_def \_par{\_nbpar\_egroup}\%
\_cs{\_printcaption\_\tmpa}\%
\_def \_cskip {\_par\_nobreak\_medskip} % space between caption and the object
\_public \caption \_cskip ;
\end{verbatim}

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

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

The \_captionsep inserts a separator between auto-generated caption number and the following caption text. Default separator is \_enspace but if the caption text starts with dot or colon, then the space is not inserted. A user can write \caption/t: My table and “Table 1.1: My table” is printed. You can redefine the \_captionsep macro if you want to use another separator.

\begin{verbatim}
\def \_printcaptiont {%
\_noindent \_wlabel\_thecapnum {\_bf\_thecaptitle~\_thecapnum}\%
\_narrowlastlinecentered\_indent \_futurelet \_next \_captionsep
}
\def \_captionsep{\_ifx \_next.\_ea \_bfnext \_else \_ifx \_next.:\_ea \_ea \_ea \_bfnext
\_else \_enspace \_fi\_fi}
\_def \_bfnext#1{{\_bf#1}}
\_let \_printcaptionf = \_printcaptiont % caption of figures = caption of tables
\end{verbatim}

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:

\begin{verbatim}
\let \_printcaptiona = \_printcaptionf \let \_everycaptiona = \_everycaptionf
\newcount \_anum \_addto\_secx \{\_anum=0 \}
\_def \_theanum {\_othe\_chapnum.\_the\_secnum.\_the\_anum}
\sdef{\_mt:a:en}{Algorithm} \sdef{\_mt:a:cs}{Algoritmus} % + your language...
\end{verbatim}

The default format of \caption text is a paragraph in block narrower by \_indent and with the last line is centered. This setting is done by the \_narrowlastlinecentered macro.

\begin{verbatim}
\def \_narrowlastlinecentered#1{%
\_leftskip=#1 plus1fil
\_rightskip=#1 plus-1fil
\_parfillskip=0pt plus2fil\_relax
}
\end{verbatim}

\_eqmark is processed in display mode (we add \_eqno primitive) or in internal mode when \_eqalignno is used (we don’t add \_eqno).

\begin{verbatim}
\optdef \_eqmark []{}{\_trylabel \_ineqmark}
\_def \_ineqmark{\_incr \_dnum
\_ifinner \_else \_eqno \_fi
\_vlabel\_thednum \_hbox{\_thednum}\%
}
\_public \eqmark ;
\end{verbatim}

The \_numberedpar ⟨letter⟩⟨name⟩ is implemented here.
\_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\_sechnum.\_the\_counterA}
\_def \_theBnum {\_othe\_chapnum.\_othe\_sechnum.\_the\_counterB}
\_def \_theCnum {\_othe\_chapnum.\_othe\_sechnum.\_the\_counterC}
\_def \_theDnum {\_othe\_chapnum.\_othe\_sechnum.\_the\_counterD}
\_def \_theEnum {\_othe\_chapnum.\_othe\_sechnum.\_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}}

\_public \_numberedpar ;

The \_printnumberedpar \theXnum {\(name\)} opens numbered paragraph and prints it. The optional parameter is in \_the\_opt. You can re-define it if you need another design.

\_printnumberedpar needs not to be re-defined if you only want to print Theorems in italic and to insert vertical skips (for example). You can do this by the following code:

\def\_theorem {\medskip\bgroup\it \numberedpar A{Theorem}}
\def\_endtheorem {\par\egroup\medskip}
\_theorem Let $M$ be... \_endtheorem

2.27 Lists, items

\_codelist \begitems (Lists: begitems, enditems <2021-03-10>) % preloaded in format

\_aboveliskip is used above the list of items,
\_belowliskip is used below the list of items and
\_interliskip is used between items.
\_listskipA is used as \_listskipamount at level 1 of items.
\_listskipB is used as \_listskipamount at other levels.
\_setlistskip sets the skip dependent on the current level of items

The \_itemnum is locally reset to zero in each group declared by \begitems. So nested lists are numbered independently. Users can set initial value of \_itemnum to another value after \begitems if they want. Each level of nested lists is indented by the new \_iindent from left. The default item mark is \_printitem.

The \begitems runs \_aboveliskip only if we are not near below a title, where a vertical skip is placed already and where the \_ Penalty 11333 is. It activates * and defines it as \_startitem.

The \enditems runs \_isnextchar\_par{}{\_noindent} thus the next paragraph is without indentation if there is no empty line between the list and this paragraph (it is similar behavior as after display math).
\newcount\_itemnum \_itemnum=0
\newtoks\_printitem
\def\_begitems{\par
\_bgroup
\_advance \_ilevel by1
\_setlistskip
\ifnum\lastpenalty<10000 \_aboveliskip \fi
\_itemnum=0 \_adef*{\_relax \_ifmmode* \_else \_ea \_startitem \_fi}
\_advance \leftskip by \_iindent
\_printitem=\_defaultitem
\_the \_everylist \_relax
}
\def\_enditems{\_par \_belowliskip \_egroup \_isnextchar \_par{}{\_noindent}}
\def\_startitem{\_par \ifnum\_itemnum>0 \_interliskip \fi
\_advance \_itemnum by1
\_the \_everyitem \_noindent \llap{\_the \_printitem} \ignorespaces}
\public \begitems \enditems \itemnum ;

\_def\novspaces{\_removelastskip \_listskipamount=0pt \_relax}
\public \novspaces ;

Various item marks are saved in \_item:{\_letter} macros. You can re-define then or define more such macros. The \_style{\_letter} does \_printitem=\_item:{\_letter} first, then \_style{\_letter} does \_printitem=\_item:{\_letter} when it is used and finally, \_startitem alias * uses \_printitem.

\_def\_style#1{\ifcsname _item:#1\_endcsname \_printitem=\_csname _item:#1\_endcsname \_relax\else \_printitem=\_defaultitem \fi}
\_sdef{_item:o}{\_raise\_4ex \_hbox{$\_scriptscriptstyle \_bullet$} }
\_sdef{_item:-}{\_hbox{\_fullrectangle{\_dimen}}} 
\_sdef{_item:n}{\_the \_itemnum. }
\_sdef{_item:N}{\_the \_itemnum) }
\_sdef{_item:i}{\_raise\_3ex \_fullrectangle{\_6ex} \_kern\_5em}
\_sdef{_item:I}{\_uppercase \_ea{\_raise\_3ex \_fullrectangle{\_6ex} \_kern\_4em} \_kern\_5em}
\_sdef{_item:a}{\_athel \_itemnum) }
\_sdef{_item:A}{\_uppercase \_ea{\_athel \_itemnum) }}
\_sdef{_item:x}{\_raise\_3ex \_fullrectangle{\_6ex} \_kern\_5em}
\_sdef{_item:X}{\_raise\_2ex \_fullrectangle{\_1ex} \_kern\_5em}
\_ate{\_num} returns the \_num's lowercase letter from the alphabet.
\_fullrectangle{(\_dimen)} prints full rectangle with given \_dimen).

\_def\_fullrectangle#1{\_hbox{\_vrule height\_1 width\_1}}
\_def\_ate#1{\ifcase#1\or a\_or b\_or c\_or d\_or e\_or f\_or g\_or h\_or
i\_or j\_or k\_or l\_or m\_or n\_or o\_or p\_or q\_or r\_or s\_or t\_or
u\_or v\_or w\_or x\_or y\_or z\_else ?\_fi}
\public \_style ;

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.
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.

\_def\_ttskip{\_medskip} % space above and below \begtt, \verbinput
\mathchardef\_ttpenalty=100 % penalty between lines in \begtt, \verbinput
\newcount\_viline % last line number in \verbinput
\newread\_vifile % file given by \verinput
\_def\_ttfont{\_tt} % default tt font

\_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⟩} in \write parameters. This protect the expansions of the \code parameter (like \, \^ etc.).

\_def\_code#1{\_unexpanded\_ea{\_csname _code _endcsname\#1}}} \_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}

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.

\_newcatcodetable \_verbatimcatcodes \_def\_setverb{\_begingroup \_def\do##1{\_catcode`##1=12 } \_dospecials \_savecatcodetable\_verbatimcatcodes % all characters are normal \_endgroup \_setverb \_def\_setverb{\_catcodetable\_verbatimcatcodes }\_let\_activettchar=\_verbchar % for backward compatibility \_public \_code \verbchar \activettchar ;

\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 \begtt 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 \begtt macro does \setverb and another preprocessing, sets \endlinechar to \textasciitilde J 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\textasciitilde J{...}

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

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

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

When \prepareverbdata finishes then \startverb runs \printverb loop over each line of the data and does a final work: last skip plus \noindent in the next paragraph.

\begin{verbatim}
126 \_def\begtt{\_par \_begingroup \_adef\%##1\_relax{\_relax}\_begtti}
127 \_eoldef \_begtti#1{\_wipeepar \_setxhsize
128 \_vskip\parskip \_ttskip
129 \_setverb
130 \_ifnum\_ttline<0 \_let\_printverblinenum=\_relax \_else \_initverblinenum \_fi
131 \_adef{\_dspa}_adef\textasciitilde J(\_parindent \_ttindent \_parskip=0pt
132 \_def\_\_bkskip \_dimezp\_tabspace\em/2\_relax}
133 \_protrudechars=0 \% disable protrusion
134 \_the\_everytt \_relax \_relax \_ttfont
135 \_def\_testcommentchars##1\_iftrue{\_iffalse}\_let\_hicomments=\_relax
136 \_savemathsb \_endlinechar=`^^J
137 \_startverb
138 }
139 \_ea\_def\_ea\_startverb \_ea##1\_csstring\endtt#2\textasciitilde J{(
140 \_prepareverbdata\_tmpb\_textasciitilde ^\textasciitilde J}{% 
141 \_ea\_printverb \_tmpb\_end
142 \_par \_restoremathsb
143 \_endgroup \_ttskip
144 \_isnextchar\_par{\_noindent}%
145 }
146 \_def\_prepareverbdata#1#2{\_def#1{#2}}
\end{verbatim}

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

The \_testcommentchars replaces the following \_iftrue to \_iffalse by default unless the \commentchars are set. So, the main body of the loop is written in the \else part of the \_iftrue condition. The \_printverbline\{(\texttt{line})\} is called here.

The \_testcommentchars\{(\texttt{line})\} expects that it starts in vertical mode and it must do \par to return the vertical mode. The \_printverbline is used here: it does nothing when \ttline<0 else it prints the line number using \llap. \_putttpenalty 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 \_printverbline is not processed in comments-blocks but we need to count the \_ttline.

\begin{verbatim}
171 \_def\_printverb #1\textasciitilde J#2{% 
172 \_ifx\_printverbline\_relax \_else \_incr\_ttline \_fi
173 \_testcommentchars \#1\relax\_relax\_relax
174 \_iftrue
175 \_ifx\_end2\_printcomments\_fi
176 \_else
\end{verbatim}
Macro \verbinput uses a file read previously or opens the given file. Then it runs the parameter scanning by \viscanparameter and \viscanminus. Finally the \doverbinput is run. At the beginning of \doverbinput, we have \viline= number of lines already read using previous \verbinput, \vinolines= the number of lines we need to skip and \vidolines= the number of lines we need to print. A similar preparation is done as in \begtt after the group is opened. Then we skip \vinolines lines in a loop a and we read \vidolines lines. The read data is accumulated into \tmpb macro. The next steps are equal to the steps done in \startverb macro: data are processed via \prepareverbdata and printed via \printverb loop.
\_global\_viline=0
\_else
\_edef\_vinolines(\_the\_tmpnum)\%
\_fi
\_vskip\_parskip\_ttskip\_wipepar\_setxhsize
\_begingroup
\_ifnum\ttline<-1\_let\_printverblinenum=\_relax\_else\_initverblinenum\_fi
\_setverb\_adef(\_dsp)\_adef\^^I\_parindent=\_ttindent\_parskip=0pt
\_edef(\_hskip\_dimexpr\_tabspaces em/2\_relax)\%
\_protrudechars=0 % disable protrusion
\_the\_everytt\_relax\_tmpb\_relax\_ttfont
\_savemathsb\_endlinechar=`^^J\_tmpnum=0
\_loop\_ifeof\vifile\_tmpnum=\_vinolines\_space\_fi
\_ifnum\_tmpnum<\_vinolines\_space\_vireadline\_advance\_tmpnum by1\_repeat %% skip lines
\_edef\_ttlinesave(\_global\_ttline=\_the\_ttline)\%
\_loop\_ifnum\ttlinesave(\_global\_ttline=\_the\_ttline)\%
\_vireadline\_loop\_ifnum\_tmpnum<\_vidolines\_space\_vireadline
\_ifnum\_vidolines=0\_else\_advance\_ttmnum by1\_fi
\_ifnum\_vidolines=0 \_else\_visaveline\_fi \% save line
\_repeat
\_ea\_prepareverbdata\_ea\_tmpb\_ea(\_tmpb--J)\%
\_catcode`\_mathsb=10 \_catcode`\%=9 % used in \commentchars comments
\_ea\_printverb\_tmpb\_end
\_ttlinesave
\_par\_restoremathsb
\_endgroup
\_ttskip
\_ismextchar\_par(\_noindent)\%
\_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 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 \%
\commentchars \#12 \catcode`\_ea\ commentchars\%
\commentchars \#13 \_ea\ commentchars\%
\commentchars\%

There is one limitation when \TeX interprets the comments declared by \commentchars. Each block of comments is accumulated to one line and then it 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 \_iffalse or \_iffalse depend-
The \_vcomments macro is not closed in this macro
\_ifx \_initunifonts \_relax \_def \_vcomments {\_ifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum % not closed in this macro
\_ifx \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum 0>0
\_ifx \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum % empty comment=\_vcomments
\_ifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum % used in \_hicomments
\_ifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum \_iifnum % default value of \_testcommentchar
\_def \_testcommentchars \_iftrue {\_iffalse} % default value of \_testcommentchar
\_def \_printcomments {\_ttskip {\_catcodetable0 \_rm \_everypar={}\_noindent \_ignorespaces \_scantextokens \_ea {\_vcomments} \_par} \_ttskip}
\_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.

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:

\begtt
\begtt
\endtt

\begtt
\begtt
\endtt

---

2.28.2 Listings with syntax highlighting

The user can write
\begtt
\begtt
\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:
OpTEX provides \texttt{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 hisicolors-⟨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 \newtoks\hicolors{⟨name⟩}(without the \_ prefix) and set the color palette there. It has precedence before \hicolors{⟨name⟩}(with the \_ prefix) declared in the hisicolors-⟨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, 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{/\*\*}}

This replaces all C comments /\*...\*/ by \x C{/\*...\*/}. But C comments may span multiple lines, i.e. the \^J 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 \z is from from a single line. Parameters not
crossing line boundary are represented by \texttt{x C\{\texttt{text}\}} and replaced by \texttt{z C\{\texttt{text}\}} without any change. But:

\[
\texttt{x C\{\texttt{text1}\}^J\texttt{text2}^J\texttt{text3}}
\]
is replaced by

\[
\texttt{z C\{\texttt{text1}\}^J\texttt{z C\{\texttt{text2}\}^J\texttt{z C\{\texttt{text3}\}}}
\]
\[
\texttt{\langle letter\rangle\{\texttt{text}\}} is expanded to \texttt{\_z:letter\{\texttt{text}\}} and if \texttt{\hicolor\langle letter\rangle\color\{\texttt{color}\}} is declared then \texttt{\_z:letter\{\texttt{color}\}} expands to \{\texttt{color}\}\{\texttt{text}\}. So, required color is activated for each line separately (e.g. for \texttt{C comments spanning multiple lines}).

• \texttt{\{\texttt{text}\}} is replaced by \texttt{\\_xscan\{\texttt{text}\}} in the post-processing phase. It should be used for macros without a parameters. You cannot use unprotected macros as replacement text before the post-processing phase, because the post-processing phase is based on the expansion of the whole verbatim text.

The macros \texttt{\replfromto} and \texttt{\replthis} manipulate the verbatim text that is already stored in the \texttt{\_tmpb} macro.

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

\texttt{\replfromto} continues by finding next \texttt{\{\texttt{from}\}}, then, next \texttt{\{\texttt{to}\}} repeatedly over the whole verbatim text. If the verbatim text ends with opening \texttt{\{\texttt{from}\}} but has no closing \texttt{\{\texttt{to}\}}, then \texttt{\{\texttt{to}\}} is appended to the verbatim text automatically and the last part of the verbatim text is replaced too.

The first two parameters are expanded before use of \texttt{\replfromto}. You can use \texttt{\csstring\%} or something else here.

The \texttt{\replthis\{\texttt{pattern}\}\{\texttt{replacement}\}} replaces each \texttt{\{\texttt{pattern}\}} by \texttt{\{\texttt{replacement}\}}. Both parameters of \texttt{\replthis} are expanded first.

The patterns \texttt{\{\texttt{from}\}}, \texttt{\{\texttt{to}\}} and \texttt{\{\texttt{pattern}\}} are not found when they are hidden in braces \{...\}. E.g.

\[
\replfromto{\texttt{/**}}{\texttt{/}*/}{\texttt{x C\{\texttt{*/}\texttt{*/}\}}}
\]

replaces all \texttt{C comments by \texttt{x C\...}. The patterns inside \{...\} are not used by next usage of \texttt{\replfromto} or \texttt{\replthis} macros.

The \texttt{\xscan} macro replaces occurrences of \texttt{x} by \texttt{z} in the post-processing phase. The construct \texttt{\x C\{\texttt{letter}\}\{\texttt{text}\}} expands to \texttt{\xscan \{\texttt{letter}\}\{\texttt{text}\}^J^J}. If \#3 is \texttt{\_end} then it signals that something wrong happens, the \texttt{\{from\}} was not terminated by legal \texttt{\{to\}} when \texttt{\replfromto} did work. We must fix this by using the \texttt{\xscanR} macro.

\[
\texttt{\_def\_xscan#1#2^J\texttt{\_tmpb\{\_ifx\_end\texttt{\_ea\_xscan\texttt{\_ea\_tmpb\_fi}}}}
\]
The \hicolor{letter} \color{text} defines \_z:{\color{text}} as \{color\}{text}. It should be used in the context of \x{letter}{text} macros.

\hisyntax{xml} re-defines default \_prepareverbdata{macro}{verbtext}, but in order to do it does more things: It saves \verbtext{} to \_tmpb, appends \_n around spaces and ^^J characters in pre-processing phase, opens hisyntax{name}.opm file if \_hisyntax{name} is not defined. Then \_the\_hisyntax{name} is processed. Finally, the post-processing phase is realized by setting appropriate values to the \x and \y macros and doing \edef{\_tmpb}{}.

\_def\_hisyntax#1{\_def\_prepareverbdata##1##2{\_let\n=\_relax \_let\b=\_relax \_def\t{\_noexpand\t\n} \_let\_start=\_relax \_adef{ }{\_noexpand\ }\_edef{\_start\_end}{\_end} \_replthis{\_start\_end}{\_end} \_def\w####1{####1}\_def\y####1####2{\_xscan{####1}####2\_end} \_def\z####1{\_cs{####1}}\_def\hskip{\_dimexpr\tabspaces em/2\_relax} \\_localcolor \_localcolor}

\_public \hisyntax \hicolor ;

Aliases for languages can be declared like this. When \hisyntax{xml} is used then this is the same as \hisyntax{html}.

2.29 Graphics

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

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

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 the \transformbox{\{transformation\}}{\copy\mypic}.
\inspic accepts old syntax \inspic ⟨filename⟩⟨space⟩ or new syntax \inspic{⟨filename⟩}. So, we need to define two auxiliary macros \_inspicA and \_inspicB.

You can include more \pdfximage parameters (like page(number)) in the \_picparams macro. All \inspic macros are surrounded in \hbox in order user can write \moveright\inspic ... or something similar.

\pdfscale{(x-scale)}{(y-scale)} and \pdfrotate{(degrees)} macros are implemented by \pdfsetmatrix primitive. We need to know the values of sin, cos function in the \pdfrotate. We use Lua code for this.
The \texttt{\textbackslash transformbox}{\{\{transformation\}\}\{\{text\}\}} is copied from OPmac trick 0046. The \texttt{\textbackslash rotbox}{\{\{degrees\}\}\{\{text\}\}} is a combination of \texttt{rotsimple} from OPmac trick 0101 and the \texttt{\textbackslash transformbox}. Note, that \texttt{\textbackslash rotbox{-90}} puts the rotated text to the height of the outer box (depth is zero) because code from \texttt{rotsimple} is processed. But \texttt{\textbackslash rotbox{-90.0}} puts the rotated text to the depth of the outer box (height is zero) because \texttt{\textbackslash transformbox} is processed.

\texttt{\textbackslash scantwodimens} scans two objects with the syntactic rule \{\{dimen\}\} and returns \{\{number\}\}{\{number\}\} in sp unit. \texttt{\textbackslash puttext}{\{\{right\}\} \{\{up\}\} \{\{text\}\}} puts the \{\{text\}\} to desired place: From current point moves \{\{down\}\} and \{\{right\}\}, puts the \{\{text\}\} and returns back. The current point is unchanged after this macro ends. \texttt{\textbackslash putpic}{\{\{right\}\} \{\{up\}\} \{\{width\}\} \{\{height\}\} \{\{image-file\}\}} does \texttt{\textbackslash puttext} with the image scaled to desired \{\{width\}\} and \{\{height\}\}. If \{\{width\}\} or \{\{height\}\} is zero, natural dimension is used. The \texttt{\textbackslash nospec} is a shortcut to such a natural dimension. \texttt{\textbackslash backgroundpic}{\{\{image-file\}\}} puts the image to the background of each page. It is used in the \texttt{\textbackslash slides} style, for example.

\texttt{\textbackslash def\{\texttt{\textbackslash scantwodimens}\}\{\}}\texttt{\{\textbackslash directlua{tex.print(string.format('{\%pcent d}{\%pcent d}',token.scan_dimen(),token.scan_dimen()))}\}}
\texttt{\textbackslash nospec} is a shortcut to such a natural dimension.
\_def\_puttext{\_ea\_ea\_ea\_puttextA\_scantwodimens}
\_def\_puttextA#1#2#3{{\_setbox0=\_hbox{{#3}}\_dimen1=#1sp \_dimen2=#2sp \_puttextB}}
\_def\_puttextB{\_ifvmode
\_ifdim\_prevdepth>\_zo \_vskip-\_prevdepth \_relax \_fi
\_nointerlineskip
\_fi
\_wd0=\_zo \_ht0=\_zo \_dp0=\_zo
\_vbox to\_zo{\_kern-\_dimen2 \_hbox to\_zo{\_kern\_dimen1 \_box0\_hss}\_vss}}
\_def\_putpic{\_ea\_ea\_ea\_putpicA\_scantwodimens}
\_def\_putpicA#1#2{\_dimen1=#1sp \_dimen2=#2sp \_ea\_ea\_ea\_putpicB\_scantwodimens}
\_def\_putpicB#1#2#3{{\_setbox0=\_hbox{\_picwidth=#1sp \_picheight=#2sp \_inspic{#3}}\_puttextB}}
\_newbox\_bgbox
\_def\backgroundpic#1{\_setbox\_bgbox=\_hbox{\_picwidth=\_pdfpagewidth \_picheight=\_pdfpageheight \_inspic{#1}}\_pgbackground={\_copy\_bgbox}}
\_def\nospec{0pt}
\_public\_puttext\_putpic\_backgroundpic;
\_circle{\langle x\rangle}{\langle y\rangle} creates an ellipse with \langle x\rangle axis and \langle y\rangle axis. The origin is in the center.
\_oval{\langle x\rangle}{\langle y\rangle}{\langle roundness\rangle} creates an oval with \langle x\rangle, \langle y\rangle size and with the given \langle roundness\rangle. The real size is bigger by 2\langle roundness\rangle. The origin is at the left bottom corner.
\_mv{\langle x\rangle}{\langle y\rangle}{\langle curve\rangle} moves current point to \langle x\rangle, \langle y\rangle, creates the \langle curve\rangle and returns the current point back. All these macros are fully expandable and they can be used in the \pdfliteral argument.
\_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}}
\_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}}

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

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

The \_circle{\langle text\rangle} is an example of \_circle usage.
The \_oval{\langle text\rangle} is an example of \_circle usage.
The \_ratio, \_width, \_color, \_color, \_shadow and \_overlapmargins are parameters, they can be set by user in optional brackets [...]. For example \_fcolor=\Red does \_let\_fcolorvalue=\Red and it means filling color.
The \_setfcolors uses the \_fcolor macro to separate filling (non-stroking) color and stroking color. The \_coc macro means “create oval or circle” and it expands to the stroking primitive S or filling primitive f or both B. Only boundary stroking is performed after \_fcolor=\relax. You cannot combine \_fcolor=\relax with \_shadow=Y.
Just before defining shadows, which require special graphics states, we define means for managing these graphics states and other PDF page resources (graphics states, patterns, shadings, etc.). Our mechanism, defined mostly in Lua (see 2.39.4, uses single dictionary for each PDF page resource type (extgstate, etc.) for all pages \( \texttt{pdfpageresources} \) just points to it.

The macro \( \texttt{\_addextgstate}\langle \texttt{PDF name} \rangle\{ \texttt{PDF dictionary} \} \) is a use of that general mechanism and shall be used for adding more graphics states. It must be used \textit{after} \( \texttt{\_dump} \). It’s general variant defined in Lua is \( \texttt{\_addpageresource\{resource type\}\langle \texttt{PDF name} \rangle\{ \texttt{PDF dictionary} \}} \). You can use \( \texttt{\_pagesresource} \) or \( \texttt{\_pagesresource} \) 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 \( \texttt{\_shadowmoveto} \) vector and with blurred boundary. A waistline with the width \( 2\times \texttt{\_shadowb} \) around the boundary is blurred. The \texttt{\_shadowlevels} levels of transparent shapes is used for creating this effect. The \( \texttt{\_shadowlevels}+1/2 \) level is equal to the shifted given path.

The \( \texttt{\_doshadow\{curve\}} \) does the shadow effect.

A generic macro \( \texttt{\_clipinpath\{x\} \{y\} \{curve\} \{text\}} \) 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 bp without the unit here. The macros \( \texttt{\_clipinoval\{x\} \{y\} \{width\} \{height\} \{text\}} \) and \( \texttt{\_clipincircle\{x\} \{y\} \{width\} \{height\} \{text\}} \) are defined here. These macros read normal TeX dimensions in their parameters.
2.30 The \table macro, tables and rules

2.30.1 The boundary declarator:

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

Each table item has its group. The \( \langle\text{cmd}\rangle \) 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:

\table{c:(\bf)c:(\Red)c}{\langle\text{data}\rangle}

2.30.2 Usage of the \tabskip primitive

The value of \tabskip primitive is used between all columns of the table. It is glue-type, so it can be stretchable or shrinkable, see next section 2.30.3.

By default, \tabskip is 0pt. It means that only \tabiteml, \tabitemr and \( \langle\text{cmds}\rangle \) 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 \( \langle\text{tabskip}\langle\text{value}\rangle \) in the \( \langle\text{declaration}\rangle \) 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 \tabskip. 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:

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

Example of \texttt{\table to\{size\}}:

\begin{verbatim}
\hbox{aaa & Ddkas jd dsjds ds cgha sfgs dd fddzf dfhz xxz \\
     dras ffg hksd kds d sdjds h sd jd dsjds ds cgha \\
     sfgs dd fddzf dfhz xxz. \hss}
\end{verbatim}

This table has its width \hsize. The first column starts at the left boundary of this table and it is justified left (to the boundary). The second column ends at the right boundary of the table and it is justified right (to the boundary). The space between them is stretchable and shrinkable to reach the given width \hsize.

Example of \texttt{\table pxto\{size\}} (means “paragraphs expanded to”):

\begin{verbatim}
\hbox{aaa & Ddkas jd dsjds ds cgha sfgs dd fddzf dfhz xxz \\
     dras ffg hksd kds d sdjds h sd jd dsjds ds cgha \\
     sfgs dd fddzf dfhz xxz. \hss}
\end{verbatim}

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 \texttt{p\{\langle coefficient\}\tsize\}} columns in the table when \texttt{pxto} keyword is used.

\begin{verbatim}
\table pxto13cm {r p{3.5\tsize} p{2\tsize} p{\tsize} l}\{\langle data\}\}
\end{verbatim}

This gives the ratio of widths of individual paragraphs in the table 3.5:2:1.

2.30.4 \texttt{\eqbox}: boxes with equal width across the whole document

The \texttt{\eqbox\{\langle label\}\{\langle text\}\}} behaves like \texttt{\hbox\{\langle text\}\}} in the first run of \TeX. But the widths of all boxes with the same label are saved to \texttt{.ref} file and the maximum box width for each label is calculated at the beginning of the next \TeX run. Then \texttt{\eqbox\{\langle label\}\{\langle text\}\}} behaves like \texttt{\hbox to \{dim:label\} \{\hss \langle text\}\hss}, where \texttt{\{dim:label\}} is the maximum width of all boxes labeled by the same \texttt{\langle label\}}. The documentation of the \LaTeX package \texttt{eqparbox} includes more information and tips.

The \texttt{\eqboxsize\{\langle label\}\{\langle dimen\}\}} expands to \texttt{\{dim:label\}} if this value is known, else it expands to the given \texttt{\langle dimen\}}.

The optional parameter r or l can be written before \texttt{\{\langle label\}\}} (for example \texttt{\eqbox r\{label\}\{text\}}) if you want to put the text to the right or to the left side of the box width.

Try the following example and watch what happens after first \TeX run and after the second one.

\begin{verbatim}
\def\leftitem#1{\par
 \noindent \hangindent=\eqboxsize\{items\}\{2em\}\hangafter=1
 \eqbox r\{items\}\{#1\}\ignorespaces}
\leftitem {\bf first} \lorem[1]
\leftitem {\bf second one} \lorem[2]
\leftitem {\bf final} \lorem[3]
\end{verbatim}

146
2.30.5 Implementation of the \table macro and friends

The result of the \table\{\{declaration\}\}\{\{data\}\} macro is inserted into \_tablebox. You can change default value if you want by \let\_tablebox=\vtop or \let\_tablebox=\relax.

We save the to\{size\} or pxto\{size\} to \_tablew and \_tablew sets the to\{size\} to the \_tablew macro. If pxto\{size\} is used then \_tablew is empty and \_tmpdim includes given \{size\}. The \_ifpxto returns true in this case.

The \table continues by reading \{\{declaration\}\} in the \_tableA macro. Catcodes (for example the | character) have to be normal when reading \table parameters. This is the reason why we use \catcodetable here.

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\{\{declaration\}\} macro scans the \{declaration\} by \scantabdata\#1\relax and continues by processing \{\{data\}\} by \_tableB. The trick \_tmptoks=\{\{data\}\}\edef\_tmpb{\_the\_tmptoks} is used here in order to keep the hash marks in the \{data\} unchanged.

The \_table\{\{data\}\} macro sets the \{data\} to \_tmpb and does \replstrings to prefix each macro \crl (etc.) by \_crcr. 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

\table \tabskip=\_tabskipmid \halign{\{converted declaration\}\table\{\{data\}\}\crcr}

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

If pxto keyword was used, then we set the virtual unit \tsize to \~\hsize first. Then the first attempt of the table is created in box 0. All columns where p{..\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 firts pass. The desired table width is stored in the \_tmpdim, so \_tmpdim-\wd0 is the rest which have to be filled by \tsize. 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 noneempty if the `to` keyword was used.

The ⟨data⟩ are re-tokenized by `\scantextokens` in order to be more robust to catcode changing inside the ⟨data⟩. But inline verbatim cannot work in special cases here like ‘{’ for example.

\_tablereplstrings replaces each \crl etc. to \crcr\crl. The reason is: we want to use macros that scan its parameter to a delimiter written in the right part of the table item declaration. The \crcr cannot be hidden in another macro in this case.

The `\scantabdata` macro converts `table`'s ⟨declaration⟩ to `\halign ⟨converted declaration⟩`. The result is stored into `\tabdata` tokens list. For example, the following result is generated when ⟨declaration⟩=|cr||cl|.

```
tabdata: \_vrule\_the\_tabiteml{\_hfil\#\_unsskip\_hfil}\_the\_tabitemr\_tabstrutA
 &\_the\_tabiteml{\_hfil\#\_unsskip}\_the\_tabitemr
 &\_vrule\_kern\_vvkern\_vrule\_tabstrutA
 &\_the\_tabiteml{\_hfil\#\_unsskip\_hfil}\_the\_tabitemr\_tabstrutA
 &\_the\_tabiteml{\_relax\#\_unsskip\_hfil}\_the\_tabitemr\_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 \crl macro.
The \_addtabitemx adds the boundary code (used between columns) to the converted declaration. This code is \texttt{\begingroup \relax} \texttt{\colnum=} \texttt{\endgroup} \texttt{\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 ⟨data⟩ item because \halign first expands the front of the item and the left part of the declaration is processed after this. Use \texttt{relax} the \texttt{\colnum} instead. Or you can write:

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

This example prints 1/3 2/3 3/3, because the value of the \texttt{\colnum} is equal to the total number of columns before left part of the column declaration is processed.

The default “declaration letters” c, l, r and p are declared by setting \texttt{\tabdeclarec, \tabdeclarer, \tabdeclarel} and \texttt{\paramtabdeclarep} macros. In general, define \texttt{\def \tabdeclare{\texttt{⟨letter⟩}{...}}} for a non-parametric letter and \texttt{\def \paramtabdeclare{\texttt{⟨letter⟩}{...}}} for a letter with a parameter. The double hash \texttt{##} 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 \texttt{##} with fills are in group. The reason can be explained by following example:

\begin{verbatim}
\table{|c|c|}{\crl \Red A & B \crl} \end{verbatim}

We don’t want vertical line after red A to be in red.
The \_tableparB initializes the paragraphs inside the table item and \_tableparC closes them. They are used in the _paramtabledeclarep macro. The first paragraph is not indented.

Users put optional spaces around the table item typically, i.e. they write & text & instead &text&. The left space is ignored by the internal T\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".

The \fS macro is more tricky. The \_tableparbox isn’t printed immediately, but _setbox2 is prefixed by the macro _tableparA, which is empty by default (used in _tablepar). The _tableparD is processed after the box is set: it checks if there is only one line and prints \hbox to\hsize{\hfil ⟨this line⟩ \hfil} in this case. In other cases, the box is printed.

The family of _cr* macros _crl, _crll, _crli, _crlli, _crlp and _tskip ⟨dimen⟩ is implemented here. The _zerotabrule is used to suppress the negative \lineskip declared by _tablinespace.
The \mspan\{\langle number\rangle\}\{\langle declaration\rangle\}\{\langle text\rangle\} macro generates similar \omit\span\omit\span\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 \star (\textwidth + \textheight + 2pt) / 2.

The #1 parameter must be a one-digit number. If you want to set more digits then use braces.

The parameters of primitive \vrule and \hrule keeps the rule “last wins”. If we re-define \hrule to \_orihrule height1pt then each usage of redefined \hrule uses 1pt height if this parameter isn’t overwritten by another following height parameter. This principle is used for settings another default rule thickness than 0.4pt by the macro \rulewidth.

The \frame\{\langle text\rangle\} uses “\vbox in \vtop” trick in order to keep the baseline of the internal text at the same level as outer baseline. User can write \frame{abcxyz} in normal paragraph line, for example and gets the expected result: abcxyz. The internal margins are set by \vvkern and \hhkern parameters.

\eqbox and \eqboxsize are implemented here. The widths of all \eqboxes are saved to the .ref file in the format \_Xeqbox\{\langle label\rangle\}\{\langle size\rangle\}. The .ref file is read again and maximum box width for each \langle label\rangle is saved to \_eqb:\langle label\rangle.
2.31 Balanced multi-columns

The code used here is documented in detail in the “\TeXbook naruby”, pages 244–246, free available, http://petr.olsak.net/tbn.html, but in Czech. Roughly speaking, macros complete all material between \begmulti⟨\num-columns⟩ and \endmulti into one \vbox. Then the macro measures the amount of free space at the current page using \pagegoal and \pagetotal and does \vsplit of \vbox to columns with a height of such free space. This is done only if we have enough amount of material in \vbox to fill the full page by columns. This is repeated in a loop until we have less amount of material in \vbox. Then we run \balancecolumns which balances the last part of the columns. Each part of printed material is distributed to the main vertical list as \hbox{⟨columns⟩} and we need not do any change in the output routine.

If you have paragraphs in \begmulti... \endmulti environment then you may say \raggedright inside this environment and you can re-assign \widowpenalty and \clubpenalty (they are set to 10000 in Op\TeX).

If you have paragraphs in \begmulti... \endmulti environment then you may say \raggedright inside this environment and you can re-assign \widowpenalty and \clubpenalty (they are set to 10000 in Op\TeX).
% split the material to more pages?
\ifdim \dimen0>\dimen1 \splitpart
\else \balancecolumns \fi % only balancing
\multiskip \egroup
}
\def \splitpart{%
\makecolumns % full page
\vskip 0pt plus 1fil minus \baselineskip \break
\ifnum\mullines<\tmpnum \dimen0=\ht6 \else \dimen0=.8\maxdimen \fi
\divide \dimen0 by \Ncols \relax
\ifx \balancecolumns\flushcolumns \advance \dimen0 by -.5\vsize \fi
\setcolsize \vsize \dimen2=\dimen1
\advance \dimen2 by -.\baselineskip
%% split the material to more pages?
\ifvoid6 \else
\ifdim \dimen0>\dimen2 \ea \ea \ea \splitpart
\else \balancecolumns % last balancing
\fi \fi
\fi \fi
}\def \balancecolumns{\bgroup %
\setbox7=\copy6 % destination height: \dimen0
\ifdim \dimen0>\baselineskip \else \dimen0=\baselineskip \fi
\vbadness=20000 \dimen6=\wd6 \dimen1=\dimen0
\def \tmp{\createcolumns
\ifvoid6 \else 
\advance \dimen1 by .2\baselineskip
\setbox6=\copy7 \ea \tmp \fi}
\tmp
\printcolumns
\egroup
\setcolsize
⟨dimen⟩ sets initial value \dimen1=⟨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 \dimen1 side by side to the \box1.
\printcolumns prints the columns prepared in \box1. The first \hbox{} moves typesetting point to
the next baseline. Next negative skip ensures that the first line from splitted columns is at this position.
\setcolsize \vsize \dimen2=\dimen1
\advance \dimen2 by -.\baselineskip
\ifvoid6 \else
\ifdim \dimen0>\dimen2 \ea \ea \ea \splitpart
\else \balancecolumns % last balancing
\fi \fi
\fi \fi
\fi \fi
\def \splitpart{%
\makecolumns % full page
\vskip 0pt plus 1fil minus \baselineskip \break
\ifnum\mullines<\tmpnum \dimen0=\ht6 \else \dimen0=.8\maxdimen \fi
\divide \dimen0 by \Ncols \relax
\ifx \balancecolumns\flushcolumns \advance \dimen0 by -.5\vsize \fi
\setcolsize \vsize \dimen2=\dimen1
\advance \dimen2 by -.\baselineskip
%% split the material to more pages?
\ifvoid6 \else
\ifdim \dimen0>\dimen2 \ea \ea \ea \splitpart
\else \balancecolumns % last balancing
\fi \fi
\fi \fi
\def \balancecolumns{\bgroup %
\setbox7=\copy6 % destination height: \dimen0
\ifdim \dimen0>\baselineskip \else \dimen0=\baselineskip \fi
\vbadness=20000 \dimen6=\wd6 \dimen1=\dimen0
\def \tmp{\createcolumns
\ifvoid6 \else 
\advance \dimen1 by .2\baselineskip
\setbox6=\copy7 \ea \tmp \fi}
\tmp
\printcolumns
\egroup
\setcolsize
⟨dimen⟩ sets initial value \dimen1=⟨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 \dimen1 side by side to the \box1.
\printcolumns prints the columns prepared in \box1. The first \hbox{} moves typesetting point to
the next baseline. Next negative skip ensures that the first line from splitted columns is at this position.
2.32 Citations, bibliography

2.32.1 Macros for citations and bibliography preloaded in the format

Registers used by \cite, \bib macros are declared here. The \bibnum counts the bibliography items from one. The \bibmark is used when \nonumcitations is set.

\_bibp expands to \bibpart/. By default, \bibpart is empty, so internal links are in the form cite:/⟨number⟩. If \bibpart is set to ⟨bibpart⟩, then internal links are cite:/⟨bibpart⟩/⟨number⟩.

\cite [[⟨label⟩,⟨label⟩,...,⟨label⟩]] manages ⟨labels⟩ using \citeA and prints ⟨⟨bib-marks⟩⟩ using \printsavedcites.
\nocite [[⟨label⟩,⟨label⟩,...,⟨label⟩]] only manages ⟨labels⟩ but prints nothing.
\rcite [[⟨label⟩,⟨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 \bib 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.

⟨⟨bib-marks⟩⟩ 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 \bib{⟨bibpart⟩}{⟨⟨number⟩⟩}. First one includes ⟨⟨number⟩⟩ and second one includes ⟨⟨nonumber⟩⟩. The \_lastbn{⟨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⟩⟩}⟨⟨number⟩⟩ and we test this case by the command \if ‹\bib{⟨⟨number⟩⟩}⟨⟨number⟩⟩. This is true when \bib{⟨⟨number⟩⟩} expands to empty. The ⟨⟨label⟩⟩ is saved by \addcitelist and \bib{⟨bibpart⟩}{⟨⟨label⟩⟩} is re-defined directly as ⟨⟨number⟩⟩.
- ⟨⟨label⟩⟩ was declared by \_Xbib and it was used previously in the document. Then we do nothing (only data to \savedcites are saved.
Because we implement possibility of more independent bibliography lists distinguished by \_bibpart, the \_addcitelist\{(label)\} macro must add the \(\langle\text{label}\rangle\) to given \_ctlst:\{\langle\text{bibpart}\rangle\}/.

When \_addcitelist is processed before \usebib, then \_citeI\{\langle\text{label}\rangle\} is added. \usebib will use this list for selecting right records from .bib file. Then \usebib sets \_ctlst:\{\langle\text{bibpart}\rangle\}/ to \_write. If \_addcitelist is processed after \usebib, then \_Xcite\{\langle\text{bibpart}\rangle/\{\langle\text{label}\rangle\}\} is saved to the .ref file. The \_Xcite creates \_ctlstB:\{\langle\text{bibpart}\rangle/\} as a list of saved \_citeI\{\langle\text{label}\rangle\}. Finally, \usebib concats boths lists \_ctlst:\{\langle\text{bibpart}\rangle/\} and \_ctlstB:\{\langle\text{bibpart}\rangle/\} in the second \TeX run.

The \(\langle\text{bib-marks}\rangle\) (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 \shordcitations 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 \shordcitations sets simply \_lastcitenum=1. The macros for \(\langle\text{bib-marks}\rangle\) printing follows (sorry, without detail documentation). They are documented in opmac-d.pdf (but only in Czech).
The \bib{[\textit{label}]} or \vibib{[\textit{label}]} =\{\textit{bib-mark}\} 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{\textit{bibpart}}{\textit{label}}{\textit{number}}{\textit{nonumber}} into .ref file immediately using \_wbib{\textit{label}}{\textit{number}}{\textit{nonumber}}. This is the core of creation of mapping from \textit{labels} to \textit{number} and \textit{nonumber}.

\_bibA and \_bibB implement the scanner of the optional argument with the \bibmark. \_bibgl is \textsc{relax} by default but \texttt{\_slides} do \texttt{\_let \_bibgl=\_global}. \_dbib{[\textit{label}]} creates destination for hyperlinks.
The \_printbib prints the bib-entry itself. You can re-define it if you want a different design. The \_printbib starts in horizontal mode after \noindent and after the eventual hyperlink destination is inserted. By default, the \_printbib sets the indentation by \hangindent and prints numeric \langle bib-marks \rangle by \llap{[\the\bibnum]} \fi. If \nonumcitations then the \_citelinkA is not empty and \langle bib-marks \rangle (\the\bibnum nor \the\bibmark) are not printed. The text of bib-entry follows. User can create this text manually using \bib command or it is generated automatically from a .bib database by \usebib command.

The vertical space between bib-entries is controlled by \_bibskip macro.

The \usebib command is implemented in usebib.opm file which is loaded when the \usebib command is used first. The usebib.opm file loads the librarian.tex for scanning the .bib files. See the section 2.32.2, where the file usebib.opm is documented.

\nobibwarning [\langle list of bib-labels \rangle] 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.

The \usebib command reads the \langle 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.

The principle “first entry wins” is used. Suppose \usebib/s (simple) local,global. If an entry with the same label is declared in local.bib and in global.bib too then the first wins. So, you can set exceptions in your local.bib file for your document.

The \_bibskip macro is defined as follows:

\def\_bibskip {\ifnum\bibnum>0 \smallskip \fi}

\_def\_bibskip {\ifnum\bibnum>0 \smallskip \fi}

\_def\_usebib {\_par \_opinput {usebib.opm} \_usebib}

\_def\_usebib {\_par \_opinput {usebib.opm} \_usebib}

\_public \nobibwarning ;

2.32.2 The \usebib command

The file usebib.opm implements the command \usebib/\langle sorttype \rangle (\langle style \rangle) \langle bibfiles \rangle where \langle 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 style \rangle is the part of the name bib-\langle style \rangle.opm of the style file and \langle 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 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.

The principle “first entry wins” is used. Suppose \usebib/s (simple) local,global. If an entry with the same label is declared in local.bib and in global.bib too then the first wins. So, you can set exceptions in your local.bib file for your document.

The bib-\langle style \rangle.opm declares entry types (like @BOOK, @ARTICLE) and declares their mandatory and optional fields (like author, title). When a mandatory field is missing in an entry in the .bib file then a warning is printed on the terminal about it. You can suppress such warnings by command \nobibwarning [\langle bib-labels \rangle], where \langle bib-labels \rangle is a comma-separated list of labels (without spaces) where missing mandatory fields will be no warned.

Old .bib files may use the obscure notation for accents like \"o\". Recommendation: convert such old files to Unicode encoding. If you are unable to do this then you can set \bibtexhook={\oldaccents}.

2.32.3 Notes for bib-style writers

The .bib files include records in the format:

\@\langle entry-type \rangle {\langle label \rangle},

\langle field-name \rangle = "\langle field-data \rangle",

... etc

see the file demo/op-biblist.bib for a real example. The \langle entry-types \rangle and \langle field-names \rangle are case insensitive.
Ancient Bib\TeX{} has read such files and has generated files appropriate for reading by Bib\TeX{}. It has worked with a set of ⟨entry-types⟩, see the www page http://en.wikipedia.org/wiki/Bib\TeX{}. The set of entry types listed on this www page is de facto the Bib\TeX{} standard. The Op\TeX{} bib style writer must “declare” all such entry types and more non-standard entry types can be declared too if there is a good reason for doing it. The word “declare” used in the previous sentence means that a bib-style writer must define the printing rules for each ⟨entry-type⟩. The printing rules for ⟨entry-type⟩ include: which fields will be printed, in what order, by what format they will be printed on (italic, caps, etc.), which fields are mandatory, which are optional, and which are ignored in .bib records.

The style writer can be inspired by two styles already done: bib-simple.opm and bib-iso690.opm. The second one is documented in detail in section 2.32.5.

The printing rules for each ⟨entry-type⟩ must be declared by \_sdef{⟨print:⟨entry-type⟩⟩} in bib ⟨style⟩.opm file. The ⟨entry-type⟩ has to be lowercase here. Op\TeX{} supports following macros for a more comfortable setting of printing rules:

- \_bprinta [⟨field-name⟩] {⟨if defined⟩} {⟨if not defined⟩}. The part ⟨if defined⟩ is executed if ⟨field-name⟩ is declared in .bib file for the entry which is currently processed. Else the part ⟨if not defined⟩ is processed. The part ⟨if defined⟩ can include the * parameter which is replaced by the value of the ⟨field-name⟩.
- \_bprintfb [⟨field-name⟩] {⟨if defined⟩} {⟨if not defined⟩}. The same as \_bprinta, but the #1 parameter is used instead *. Differences: #1 parameter can be used more than once and can be enclosed in nested braces. The * parameter can be used at most once and cannot be enclosed in braces. Warning: if the \_bprintf commands are nested (\_bprintf in \_bprintf), then you need to write the #1*1 parameter for internal \_bprintf. But if \_bprintf commands are nested then the parameter is not duplicated.
- \_bprintc \macro {⟨if non-empty⟩}. The ⟨if non-empty⟩ part is executed if \macro is non-empty. The * parameter can be used, it is replaced by the \macro.
- \_bprintv [⟨field1⟩,⟨field2⟩,...] {⟨if defined⟩} {⟨if not defined⟩}. The part ⟨if defined⟩ is executed if ⟨field1⟩ or ⟨field2⟩ or ... is defined, else the second part ⟨if not defined⟩ is executed. There is one file name or the list field names separated by commas. The parts cannot include any parameters.

There are two special field-names: !author and !editor. The processed list of authors or editors are printed here instead of raw data, see the commands \_authorname and \_editorname below.

The bib-style writer can define \_print:BEGIN and/or \_print:END. They are executed at the beginning or end of each ⟨entry-type⟩. The formatting does not solve the numbering and paragraph indentation of the entry. This is processed by \_print:bib 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 \use\bib 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 \_stext{bib.⟨identifier⟩}, where ⟨identifier⟩ is an identifier of the phrase and the phrase itself is defined by \_sdef{mt:bib.⟨identifier⟩:⟨language⟩}{⟨phrase⟩}. See section 2.37.2 for more detail. Phrases for ⟨identifiers⟩: and, etal, edition, citedate, volume, number, prepages, postpages, editor, editors, available, availablealso, 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.

158
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 txdoc librarian for more information about it.

We want to ignore \errmessage and we want not to create \jobname.lbr file.

The \usebib command.

The \usebib.opm macro file loaded when \usebib is used
Corrections in librarian macros.

\_tmpnum=\_catcode`\@  \_catcode`\@=11
\_def\lb@checkmissingentries#1,{% we needn't \errmessage here, only \opmacwarning
\_\unless\_ifx\lb@temp\lb@eoe
\_ifcs{#1}{fields}%
\_ea\lb@checkmissingentries
\_\fi
\_def\lb@readentry#1#2#3,{% space before key have to be ignored
\_def\lb@temp{#2#3}% we need case sensitive keys
\lb@ifcs\lb@temp{requested}%{\lb@entrykey}{% fields}
\_let\lb@entrykey\lb@temp
\lb@ifcs\lb@entrykey{fields}{}%{\lb@defcs\lb@entrykey{fields}{}%\_lowercase{\lb@addfield{entrytype}{#1}}%}{\lb@next}{\lb@analyzeentry}}%{\lb@next}
\_let\lb@compareA=\lb@compare
\_let\lb@preparesortA=\lb@preparesort
\_def\lb@compare#1\lb@eoe#2\lb@eoe{% SpecialSort:
\_ifx\lb@sorttype\lb@namestring
\_ifx\_sortfield\_undefined \lb@compareA#1\lb@eoe#2\lb@eoe
\_else
\_ea\_RetrieveFieldInFor\_ea{\_sortfield}\lb@entrykey\lb@temp
\_ifx\lb@temp\_empty \_toks1={#1\lb@eoe}\_else \_toks1=\_ea{\lb@temp\lb@eoe}\_fi
\_ea\_RetrieveFieldInFor\_ea{\_sortfield}\lb@currententry\lb@temp
\_ifx\lb@temp\_empty \_toks2={#2\lb@eoe}\_else \_toks2=\_ea{\lb@temp\lb@eoe}\_fi
\_edef\lb@temp{\noexpand\lb@compareA\_space\_the\_toks1 \_space\_the\_toks2}\lb@temp
\_fi
\_else \lb@compareA#1\lb@eoe#2\lb@eoe \_fi}{\lb@next}
\_def\lb@preparesort#1#2\lb@eoe{% SpecialSort:
\_if#1-%\_def\lb@sorttype{#2}{}
\_else\_def\lb@sorttype{#1#2}{}
\_fi\lb@preparesortA#1#2\lb@eoe
\_def\lb@compare\#1\lb@eoe#2\lb@eoe{% SpecialSort:
\_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
\_bibtex\_advanced\_bibnum by1
\_isdefined{\bin:\_bibtex\_the\_bibnum}\_iftrue
\_edef\tmpb{\_csname \_bim:\_bibp\_the\_bibnum\_endcsname}{}
\_bibrmark=\_ea{\_tmpb}{}
\_isdefined{\_bim:\_bibp\_the\_bibnum\_endcsname}\_iftrue
\_csname \_bim:\_bibp\_the\_bibnum\_endcsname\_endcsname
\_if\_bibrmark=\_ea\_tmpb\_fi
\_else \_bibrmark=0\_\fi
\_edef\tmpb{\EntryKey}{}
\_isdefined{\EntryKey}\_iftrue
\_csname \_print:BEGIN\_endcsname
\_isdefined{\EntryKey}\_iftrue
\_csname \_print:\EntryKey\_endcsname
\_else
\_csname \_print:misc\_endcsname
\_fi\_fi
\_RetrieveFieldIn(entrytype) entrytype
\_csname \_print:BEGIN\_endcsname
\_csname \_print:BEGIN\_endcsname
\_isdefined{\_entries\_endcsname}\_iftrue
\_csname \_print:BEGIN\_endcsname
\_else
\_csname \_print:misc\_endcsname
\_fi\_fi
The \_bprinta, \_bprintb, \_bprintc, \_bprintv commands used in the style files:

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 name{=}value, or name{=}_{value}. No matter which form is used. If the value is pure numeric then you can say simply name{=}value. Warning: the comma after each field value is mandatory! If it is missing then the next field is ignored or badly interpreted.

The entry names and field names are case insensitive. If there exists a data field no mentioned here then it is simply ignored. You can use it to store more information (abstract, for example).

There are “standard fields” used in ancient bib\TeX{} (author, title, editor, edition, etc., see \url{http://en.wikipedia.org/wiki/BibTeX}). The iso690 style introduces several “non-standard” fields: ednote, numbering, isbn, issn, doi, url, citedate, key, bibmark. They are documented here.

Moreover, there are two optional special fields:

- lang = language of the entry. The hyphenation plus autogenerated phrases and abbreviations will be typeset by this language.
- option = options by which you can control a special printing of various fields.

There can be only one option field per each entry with (maybe) more options separated by spaces. You can declare the global option(s) in your document applied for each entry by \texttt{\biboptions={...}}.

The author field

All names in the author list have to be separated by “ and ”. Each author can be written in various formats (the von part is typically missing):

- Firstname(s) von Lastname
- von Lastname, Firstname(s)
- von Lastname, After, Firstname(s)

Only the Lastname part is mandatory. Examples:

- Petr Olšák
- Olšák, Petr
- Leonardo Piero da Vinci
- da Vinci, Leonardo Piero
- da Vinci, painter, Leonardo Piero

The separator “ and ” between authors will be converted to comma during printing, but between the semifinal and final author the word “and” (or something different depending on the current language) is printed.

The first author is printed in reverse order: “LASTNAME, Firstname(s) von, After” and the other authors are printed in normal order: “FirstName(s) von LASTNAME, After”. This feature follows the ISO 690 norm. The Lastname is capitalized using uppercase letters. But if the \texttt{\caps} font modifier is defined, then it is used and printed \texttt{\caps\_rm\_Lastname}.

You can specify the option \texttt{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-al. is appended to the list of printed authors. This text is printed only if the \texttt{aumax} value is less than the real number of authors. If you have the same number of authors in the .bib file as you need to print but you want to append et-al. then you can use \texttt{auetal} option.

There is an \texttt{aumin:\langle number\rangle} option which denotes the definitive number of printed authors if the author list is not fully printed due to \texttt{aumax}. If \texttt{aumin} is unused then \texttt{aumax} authors are printed in this case.

All authors are printed if \texttt{aumax:\langle number\rangle} option isn’t given. There is no internal limit. But you can set the global options in your document by setting the \texttt{\biboptions} tokens list. For example:

\texttt{\biboptions={aumax:7 aumin:1}}

% if there are 8 or more authors then only the first author is printed.

Examples:
author = "John Green and Bob Brown and Alice Black",
output: GREEN, John, Bob BROWN, and Alice BLACK.

author = "John Green and Bob Brown and Alice Black",
option = "aumax:1",
output: GREEN, John et al.

author = "John Green and Bob Brown and Alice Black",
option = "aumax:2",
output: GREEN, John, Bob BROWN et al.

author = "John Green and Bob Brown and Alice Black",
option = "aumax:3",
output: GREEN, John, Bob BROWN, and Alice BLACK.

author = "John Green and Bob Brown and Alice Black",
option = "auetal",
output: GREEN, John, Bob BROWN, Alice BLACK et al.

If you need to add a text before or after the author’s list, you can use the auprint: {⟨value⟩} option. The ⟨value⟩ will be printed instead of the authors list. The ⟨value⟩ can include \AU macro which expands to the authors list. Example:

author = "Robert Calbraith",
option = "auetal auprint:{\AU \space [pseudonym of J. K. Rowling]}",
output: CALBRAITH Robert [pseudonym of J. K. Rowling].

You can use the autrim:{⟨number⟩} option. All Firstnames of all authors are trimmed (i. e. reduced to initials) iff the number of authors in the author field is greater than or equal to ⟨number⟩. There is an exception: autrim:0 means that no Firstnames are trimmed. This is the default behavior. Another example: autrim:1 means that all Firstnames are trimmed.

author = "John Green and Bob Brown and Alice Black",
option = "auetal autrim:1",
output: GREEN, J., B. BROWN, A. BLACK et al.

If you need to write a team name or institution instead of authors, replace all spaces by \␣ in this name. Such text is interpreted as Lastname. You can add the secondary name (interpreted as Firstname) after the comma. Example:

author = "Czech Technical University in Prague, Faculty of Electrical Engineering",
output: CZECH TECHNICAL UNIVERSITY IN PRAGUE, Faculty of Electrical Engineering.

The editor field

The editor field is used for the list of the authors of the collection. The analogous rules as in author field are used here. It means that the authors are separated by “and”, the Firstnames, Lastnames, etc. are interpreted and you can use the options edmax:{⟨number⟩}, edmin:{⟨number⟩}, edetal, edtrim:{⟨number⟩} and edprint:{⟨value⟩} (with \ED macro). Example:

editor = "Jan Tomek and Petr Karas",
option = "edprint:\ED, editors. edtrim:1",
output: J. TOMEK and P. KARAS, editors.

If edprint option is not set then \ED, ed. or \ED, eds. 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.

ednote = "Illustrations by Robert AGARWAL, edited by Tom NOWAK)"
output: Illustrations by Robert AGARWAL, edited by Tom NOWAK.
The `\upper` command has to be used for Lastnames in the ednote field.

**The title field**
This is the title of the work. It will be printed (in common entry types) by italics. The ISO 690 norm declares, that the title plus optional subtitle are in italics and they are separated by a colon. Next, the optional secondary title has to be printed in an upright font. This can be added by `titlepost:` \{(value)\}. Example:

```
title = "The Simple Title of The Work",
or
title = "Main Title: Subtitle",
or
title = "Main Title: Subtitle",
option = "titlepost:{Secondary title}'",
```

The output of the last example: *Main Title: Subtitle*. Secondary title.

**The edition field**
This field is used only for second or more edition of cited work. Write only the number without the word "edition". The shortcut "ed." (or something else depending on the current language) is added automatically. Examples:

```
edition = "Second",
edition = "2nd",
edition = "2$^\mathrm{nd}$",
edition = "2.",
```

Output of the last example: 2. ed.

```
edition = "2."
lang = "cs",
```

Output: 2. vyd.

Note, that the example `edition = "Second"` may cause problems. If you are using language "cs" then the output is bad: Second vyd. But you can use `editionprint:` \{(value)\} option. The the \{(value)\} is printed instead of edition field and shortcut. The edition field must be set. Example:

```
edition = "whatever",
optio = "editionprint:{Second full revised edition}'",
```


You can use `\EDN` macro in `editionprint` value. This macro is expanded to the edition value. Example:

```
edition = "Second",
optio = "editionprint:{\EDN}space full revised edition}'",
or
edtion = "Second full revised edition",
optio = "editionprint:{\EDN}'",
```

**The address, publisher, year fields**
This is an anachronism from ancient BibTeX (unfortunately no exclusive) that the address field includes only the city of the publisher’s residence. No more data are here. The publisher field includes the name of the publisher.

```
address = "Berlin",
publisher = "Springer Verlag",
year = 2012,
```


Note, that the year needn’t to be inserted into quotes because it is pure numeric.

The letter a, b, etc. are appended to the year automatically if two or more subsequent entries in the bibliography list are not distinct by the first author and year fields. If you needn’t this feature, you can use the `noautoletters` option.
You can use "yearprint:{value}" option. If it is set then the {value} is used for printing year instead the real field value. The reason: year is sort sensitive, maybe you need to print something else than only sorting key. Example:

```plaintext
eyear = 2000,
option = "yearprint:{© 2000}";
```


```plaintext
eyear = "2012a",
option = "yearprint:{2012}";
```


The address, publisher, and year are typically mandatory fields. If they are missing then the warning occurs. But you can set unpublished option. Then this warning is suppressed. There is no difference in the printed output.

**The url field**

Use it without `\url` macro, but with `http://` prefix. Example:

```plaintext
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:

- Available from [http://petr.olsak.net/opmac.html](http://petr.olsak.net/opmac.html).
- Dostupné z [http://petr.olsak.net/opmac.html](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:

```plaintext
citedate = "2004/05/21",
```

Output when `en` is current: [cit. 2004-05-21].
Output when `cs` is current: [vid. 21. 5. 2004].

**The howpublished field**

This declares the available medium for the cited document if it is not in printed form. Alternatives: online, CD, DVD, etc. Example:

```plaintext
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:

```plaintext
volume = 31,
number = 3,
pages = "37--42",
```


```plaintext
volume = 31,
number = 3,
pages = "37--42",
lang = "cs",
```

Output: ročník 31, č. 3, s. 37–42.

If you disagree with the default prefixes, you can use the numbering field. When it is set then it is used instead of volume, number, pages fields and instead of any mentioned prefixes. The numbering can include macros `\VOL`, `\NO`, `\PP`, which are expanded to the respective values of fields. Example:
volume = 31,
number = 3,
pages = "37--42"
numbering = "Issue-VOL/NO, pages-PP",

Output: Issue 31/3, pages 37–42

Note: The volume, numbers, and pages fields are printed without numbering filled only in the \@ARTICLE entry. It means, that if you need to visible them in the \@INBOOK, \@INPROCEEDINGS etc. entries, then you must use the numbering field.

Common notes about entries

The order of the fields in the entry is irrelevant. We use the printed order in this manual. The exclamation mark (!) denotes the mandatory field. If the field is missing then a warning occurs during processing.

If the unpublished option is set then the fields address, publisher, year, isbn, and pages are not mandatory. If the nowarn option is set then no warnings about missing mandatory fields occur.

If the field is used but not mentioned in the entry documentation below then it is silently ignored.

• The \@BOOK entry
  
  This is used for book-like entries.
  
  Fields: author(!), title(!), howpublished, edition, ednote, address(!), publisher(!), year(!), citedate, series, isbn(!), doi, url, note.
  
  The ednote field here means the secondary authors (illustrator, cover design etc.).

• The \@ARTICLE entry
  
  This is used for articles published in a journal.
  
  Fields: author(!), title(!), journal(!), howpublished, address, publisher, month, year, [numbering or volume, number, pages(!)], citedate, issn, doi, url, note.
  
  If the numbering is used then it is used instead volume, number, pages.

• The \@INBOOK entry
  
  This is used for the part of a book.
  
  Fields: author(!), title(!), booktitle(!), howpublished, edition, ednote, address(!), publisher(!), year(!), numbering, citedate, series, isbn or issn, doi, url, note.
  
  The author field is used for author(s) of the part, the editor field includes author(s) or editor(s) of the whole document. The pages field specifies the page range of the part. The series field can include more information about the part (chapter numbers etc.).

  The \@INPROCEEDINGS and \@CONFERENCE entries are equivalent to \@INBOOK entry.

• The \@THESIS entry
  
  This is used for the student’s thesis.
  
  Fields: author(!), title(!), howpublished, address(!), school(!), month, year(!), citedate, type(!), ednote, doi, url, note.
  
  The type field must include the text “Master’s Thesis” or something similar (depending on the language of the outer document).

  There are nearly equivalent entries: \@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 \@MICS 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 \nonumcitations is set

When \nonumcitations is set then \cite prints text-oriented bib-marks instead of numbers. This style file auto-generates these marks in the form “Lastname of the first author, comma, space, the year” if the bibmark field isn’t declared. If you need to set an exception from this common format, then you can use bibmark field.

The OPmac trick http://petr.olsak.net/opmac-tricks-e.html#bibmark describes how to redefine the algorithm for bibmark auto-generating when you need the short form of the type [Au13].

Sorting

If \usebib/c is used then entries are sorted by citation order in the text. If \usebib/s is used then entries are sorted by “Lastname, Firstname(s)” of the first author and if more entries have this value equal, then the year is used (from older to newer). This feature follows the recommendation of the ISO 690 norm.

If you have the same authors and the same year, you can control the sorting by setting years like 2013, 2013a, 2013b, etc. You can print something different to the list using yearprint{⟨value⟩} option, see the section about address, publisher, and year above. The real value of year field (i.e. not yearprint value) is also used in the text-oriented bib-marks when \nonumcitations is set.

If you have some problems with name sorting, you can use the hidden field key, which is used for sorting instead of the “Lastname Firstname(s)” of authors. If the key field is unset then the “Lastname Firstname(s)” is used for sorting normally. Example:

```
author = "Světla Čmejrková",
key = "Czzmejrkova Svetla",
```

This entry is now sorted between C and D.

The norm recommends placing the auto-citations at the top of the list of references. You can do this by setting key=⟨"@"⟩, to each entry with your name because the @ character is sorted before A.

Languages

There is the language of the outer document and the languages of each entry. The ISO 690 norm recommends that the technical notes (the prefix before URL, the media type, the “and” conjunction between the semifinal and final author) maybe printed in the language of the outer document. The data of the entry have to be printed in the entry language (edition ed./vyd., Vol./ročník, No./č. etc.). Finally, there are the phrases independent of the language (for example In:). Unfortunately, the bib\TeX supposes that the entry data are not fully included in the fields so the automaton has to add some text during processing (“ed.”, “Vol.”, “see also”, etc.). But what language has to be chosen?

The current value of the \language register at the start of the .bib processing is described as the language of the outer document. This language is used for technical notes regardless of the entry language. Moreover, each entry can have the lang field (short name of the language). This language is used for ed./vyd., vol./ročník, etc. and it is used for hyphenation too. If the lang is not set then the outer document language is used.

You can use \_Mtext{bib⟨identifier⟩} if you want to use a phrase dependent on outer document language (no on entry language). Example:

```
howpublished = "\_Mtext{bib.blue-ray}" 
```

Now, you can set the variants of bib.blue-ray phrase for various languages:

```
\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
```
2.32.6 Implementation of the \bib-iso690 style

Option field.

Formating of Author/Editor lists.
\ifx\maybeetal\empty \bibconjunctionand\else , \fi
\else , \fi
\otherauthorformat
\fi
\}
\def\authormaxempty{\ifnum\NameCount>0\namecount\else \commonname \fi}
\def\authorminempty{\ifnum\NameCount=0\namecount\relax \maybeetal\fi}
\def\authormax{\ifnum\NameCount=0\namecount\relax \atitle\fi}
\def\autrim{\ifnum0\namecount<\autrim\relax \AbbreviateFirstname \fi}
\def\maybeetal{}
\ifx\upper\undefined
\ifx\caps\undefined \def\upper{\uppercase\ea}\else
\def\upper#1{{\caps\rm #1}}\fi
\fi
\let\upper=\upper
Preparing bib-mark (used when \numcitations is set).
\def\setbibmark{\ifx\dobibmark\undefined \def\dobibmark{}\fi
\RetrieveFieldIn{bibmark}\tmp
\ifx\tmp\empty \RetrieveFieldIn{year}\tmp \edef\tmp{\dobibmark, \tmp}\fi
\bibmark=\ea{\tmp}\}
Setting phrases.
\def\bibconjunctionand{\Mtext{bib.and}}
\def\preurl{\Mtext{bib.available}}
\let\predoi=\preurl
\def\postedition{\mtext{bib.edition}}
\def\Inclause{In:~}
\def\prevolume{\mtext{bib.volume}}
\def\prenumber{\mtext{bib.number}}
\def\prepages{\mtext{bib.prepages}}
\def\posteditor{\ifnum0\namecount>1 \Mtext{bib.editors}\else \Mtext{bib.editor}\fi}
\Mtext{⟨identifier⟩} expands to a phrase by outer document language (no entry language).
Non-standard field names.

\_CreateField {ednote}
\_CreateField {citedate}
\_CreateField {numbering}
\_CreateField {isbn}
\_CreateField {issn}
\_CreateField {doi}
\_CreateField {url}
\_CreateField {bibmark}

Sorting.

\_SortingOrder{name,year}{lfvj}
\_SpecialSort {key}

Supporting macros.

\_def\_bibwarning\{\_bibwarning\}
\_def\_bibwarningb\{\_bibwarning\}
\_def\_docitedate #1/#2/#3/#4\_relax{\_Mtext{bib.citedate}\_if^#2^#1\_else\_if^#3^#1/#2\_else\_cs{\_cs\_lan:\_the\_documentlanguage}dateformat\#1/#2/#3\relax\_fi\_fi}
\_def\_doyear#1{\_biboptionvalue{yearprint}\_yearprint\_ifx\_yearprint\_empty\_else\_def\YEAR{#1}\_yearprint\_fi}
\_def\_preparenumbering{\_def\VOL{\_RetrieveField{volume}}\_def\NO{\_RetrieveField{number}}\_def\PP{\_RetrieveField{pages}}}
\_def\_prepareednote{\_def\EDN{\_RetrieveField{edition}}\_def\ADDR{\_RetrieveField{address}}\_def\PUBL{\_RetrieveField{publisher}}\_def\YEAR{\_RetrieveField{year}}\_def\AU{\_bprintb[!author]{\_doauthor0{####1}}{}}\_def\ED{\_bprintb[!editor]{\_doeditor0{####1}}{}}\_preparenumbering}
\_def\_doedition#1{\_biboptionvalue{editionprint}\_editionprint\_ifx\_editionprint\_empty\_else\_def\ED{#1}\_editionprint\_fi}
\_def\_doauthor#1#2{\_prepareauedoptions{au}\_let\_iseditorlist=\_undefined\_if1#1\_def\AU{#2}\_else\_let\_authorprint=\_empty\_fi\_ifx\_authorprint\_empty\_else\_authorprint\_fi}
\_def\_doeditor#1#2{\_prepareauedoptions{ed}\_let\_firstauthorformat=\_otherauthorformat\_if#1\_def\ED{#2}\_else\_let\_authorprint=\_empty\_fi\_ifx\_authorprint\_empty\_else\_authorprint\_fi}

Entry types.

\_sdef{_print:BEGIN}{%}
\_readbiboptions
\_biboptionvalue{titlepost}\_titlepost
\_isbiboption{unpublished}\_iftrue\_let\_bibwarninga=\_relax\_let\_bibwarningb=\_relax\_fi
\_isbiboption{nowarn}\_iftrue\_let\_bibwarninga=\_relax\_let\_bibwarningb=\_relax\_fi
\_isbiboption{urlalso}\_iftrue\_def\preurl{\_Mtext{bib.availablealso}}\_fi
\_RetrieveFieldIn{lang}\_langentry\_setlang\_langentry
\_bprinta [note] {*.}{}
\_sdef{_print:END}{%}
\_bprinta [note] {(*.){%}
2.33 Sorting and making Index

\makeindex implements sorting algorithm at \TeX\ macro-language level. You need not any external program. The sorting can be used for various other applications, see an example in Op\TeX\ trick 0068.

There are two passes in the sorting algorithm. The primary pass does not distinguish between a group of letters (typically non-accented and accented). If the result of comparing two string is equal in primary pass then the secondary pass is started. It distinguishes between variously accented letters. Czech rules, for example, says: not accented before dieresis before acute before circumflex before ring. At less priority: lowercase letters must be before uppercase letters.

The \\texttt{\_sortingdatalatin} implements these rules for the languages with latin alphabets. The groups between commas are not distinguished in the first pass. The second pass distinguishes all characters mentioned in the \\texttt{\_sortingdatalatin} (commas are ignored). The order of letters in the \_sortingdatalatin macro is significant for the sorting algorithm.
Characters to be ignored during sorting are declared in \_ignoredcharsgeneric. These characters are ignored in the first pass without additional condition. All characters are taken into account in the second pass: ASCII characters with code < 65 are sorted first if they are not mentioned in the \_sortingdata... macro. Others not mentioned characters have undefined behavior during sorting.

Sorting is always processed by rules of a given language. The macros \_sortingdata\langle lang-tag\rangle, \_ignoredchars\langle lang-tag\rangle and \_compoundchars\langle lang-tag\rangle 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 lang-data.opm file (see section 2.37.4). The \_compoundchars\langle lang-tag\rangle 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 x:y in the \_compoundchars\langle lang-tag\rangle 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\langle lang-tag\rangle 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 \texttt{u < ü} in \texttt{\_sortingdata} and because \texttt{\_xcompoundcharsde} is empty, we have \texttt{Mueller < Müller} after second pass of the sorting.

You can declare these macros for more languages if you wish to use \texttt{\makeindex} with sorting rules with respect to your language. Note: if you need to map compound characters to a character, don’t use \texttt{\^{}}I, \texttt{\^{}}J or \texttt{\^{}}M because these characters have very specific category codes.

If you created \texttt{\_sortingdata} etc. for your language, please, send them to me. I am ready to add them to the file \texttt{lang-data.opm} in a new \TeX{} 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: \texttt{cote < côte < coté < côté}. This rule can be activated if the control sequence \texttt{\_secondpass\langle\_lang-tag\rangle} is set to \texttt{\_reversewords}. For example, \texttt{lang-data.opm} declares \texttt{\let\_secondpassfr=\_reversewords}.

Preparing to primary pass is performed by the \texttt{\_setprimarysorting} macro implemented here. The \texttt{\langle\_lang-tag\rangle} is saved to the \texttt{\_sortinglang} macro when sorting is initialized in \texttt{\_dosorting} (it is typically derived from current \texttt{\language} value). The \texttt{\_setprimarysorting} is called from \texttt{\_dosorting} macro and all processing of sorting is in a group. It sets actual \texttt{\_sortingdata}, \texttt{\_compoundchars} and \texttt{\_ignoredchars} if given language declares them. If not then warning will be printed using \texttt{\_nold} macro and English data are used. The \texttt{lccode} of all characters from \texttt{\_sortingdata} and \texttt{\_ignoredchars} are set. The sorted words will be converted using \texttt{\_compoundchars} followed by \texttt{\_lowercase} before first pass is run.

Preparing to secondary pass is implemented by the \texttt{\_setsecondarysorting} macro.

Strings to be sorted are prepared in \texttt{\langle\string\rangle} control sequences (to save \TeX{} memory). The \texttt{\_preparesorting \langle\string\rangle} converts \texttt{\langle\string\rangle} to \texttt{\_tmpb} with respect to the data initialized in \texttt{\_setprimarysorting} or \texttt{\_setsecondarysorting}. The compound characters are converted by the \texttt{\_docompound} macro.
Macro \_isAleB \langle string1 \rangle, \langle string2 \rangle returns the result of comparison of given two strings to \_ifAleB control sequence. Usage: \_isAleB \langle string1 \rangle, \langle string2 \rangle \_ifAleB \_else \_fi The converted strings (in respect of the data prepared for first pass) must be saved as values of \_string macros. The reason is speed: we don’t want to convert them repeatedly in each comparison. The macro \_testAleB \langle converted-string1\rangle&\relax(\converted-string2)\&\relax \_ifAleB \_else \_fi 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 \langle string1 \rangle, \langle string2 \rangle is run if the words are equal in the primary pass. It runs \_setSecondarySorting if it was not initialized already. Then prepares compared words to \_tmpa and \_tmpb and corrects them by \_prepSecondPass if needed. Finally, the test is recursively done by the macro \_testAleBSecondaryX \langle converted-string1\rangle0\_relax\langle converted-string2\rangle1\_relax

Merge sort is very effectively implemented by \TeX{} macros. The following code is created by my son Miroslav. The \_mergesort macro expects that all items in \_iilist are separated by a comma when it starts. It ends with sorted items in \_iilist without commas. So \_dosorting macro must prepare commas between items.
\_edef\_iilist{\_ea}\_ea\_mergesort\_iilist#1#2,#3}\%
\_fi\fi % zatriduji: p1+neco1,p2+neco2, (#1#2=p1+neco1 #3=p2)
\_isAleB #1#3\_ifAleB % p1<p2
\_addto\_iilist(#1)% % p1 do bufferu
\_sortreturn{\_fif\_mergesort#2,#3}% % \mergesort neco1,p2+neco2,
\_else % p1>p2
\_addto\_iilist(#3)% % p2 do bufferu
\_sortreturn{\_fif\_mergesort#1#2,}% % \mergesort p1+neco1,neco2,
\_fi
\_relax % zarazka, na ktere se zastavi \sortreturn
\_def\_sortreturn#1#2\_fi\_relax{#1} \_def\_fif\_fi
\_def\_gobbletoend #1\_end{}

The \dosorting \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 (c) in \langle c\rangle\langle string\rangle should be whatever. It does not influence the sorting. OpTeX uses comma at this place for sorting indexes: \langle word1\rangle \langle word2\rangle \langle word3\rangle ... .

The current language (chosen for hyphenation patterns) is used for sorting data. If the macro \_sortinglang is defined as \langle lang-tag\rangle (for example \_def\_sortinglang{de} for German) then this has precedence and current language is not used. Moreover, if you specify \_asciisortingtrue then ASCII sorting will be processed and all language sorting data will be ignored.

French rules needs reverse reading the words in the second pass. The \_reversewords is activated in this case and adds new job to the macro \_prepsortsecondpass: it reverses the letters in the compared words (saved in \_tmpa and \_tmpb) by the expandable \_sortrevers macro. The \_prepsortsecondpass 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` \texttt{\{word\}} prints one item to the index. If \texttt{\{word\}} is defined then this is used instead real \texttt{\langle word\rangle} (this exception is declared by `\iiis` macro). Else \texttt{\langle word\rangle} is printed by `\_printii`. Finally, \texttt{\_printiipages} prints the value of \texttt{\{word\}}, i.e. the list of pages.

\texttt{\_printii} \texttt{\langle word\rangle} & does more intelligent work because we are working with words in the form \texttt{\langle main-word\rangle/\langle sub-word\rangle/\langle sub-sub-word\rangle}. The `\everyii` tokens register is applied before `\noindent`. User can declare something special here.

The `\_newiiletter\{\langle letter\rangle\}` macro is empty by default. It is invoked if first letter of index entries is changed. You can declare a design between index entries here. You can try, for example:

\begin{verbatim}
\def\_newiiletter#1#2{%
  \bigskip \hbox{\setfontsize{at15pt}\bf\uppercase{#1}}\medskip}
\end{verbatim}

\texttt{\_printiipages} \texttt{\langle pglist\rangle \&} gets \texttt{\langle pglist\rangle} in the form \texttt{\langle pg\rangle:\langle type\rangle,\langle pg\rangle:\langle type\rangle,\ldots,\langle pg\rangle:\langle type\rangle} and it converts them to \texttt{\langle pg\rangle, \langle pg\rangle, (from) \sim (to), \langle pg\rangle} etc. The same pages must be printed only once and continuous consequences of pages must be compressed to the form \texttt{(from)-(to)}. Moreover, the consequence is continuous only if all pages have the same \texttt{\langle type\rangle}. Empty \texttt{\langle type\rangle} is most common, pages with \texttt{b \langle type\rangle} must be printed as bold and with \texttt{i \langle type\rangle} as italics. Moreover, the \texttt{\langle pg\rangle} mentioned here are \texttt{\langle gpageno\rangle}, but we have to print \texttt{\langle pageno\rangle}. The following macros solve these tasks.
You can re-define \_pgprint \{\langle gpageno\rangle:\{\langle iitype\rangle\}\} if you need to implement more \langle iitypes\rangle.

The \_Xindex{\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 \_Xindex.

The implementation of macros \_ii, \_iid, \_iis follows. Note that \_ii works in the horizontal mode in order to the \write whatsis is not broken from the following word. If you need to keep vertical mode, use \iindex{\langle word\rangle} \_iitype {\langle type\rangle} is saved \_\iitype {\langle type\rangle} to the \_iitiesaved macro. It is used in the \iindex macro.
2.34 Footnotes and marginal notes

\gfnotenum is a counter which counts footnotes globally in the whole document.
\lfnotenum is a counter which counts footnotes at each chapter from one. It is used for local page footnote counters too.
\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

\fnotenumpages
\def \printfnotemark {
  \ifcase \fnotenum
  *or**or***or$^\mathbox{†}$or$^\mathbox{‡}$or$^\mathbox{††}$or$^\mathbox{‡‡}$or$^\mathbox{†††}$\fi
}

This code gives footnotes* and ** and*** and† etc. and it supposes that there are no more than 6 footnotes at one page.

If you want to distinguish between footnote marks in the text and in the front of the footnote itself, then you can define \printfnotemarkA and \printfnotemarkB.

The \fnotelinks⟨colorA⟩⟨colorB⟩ implements the hyperlinked footnotes (from text to footnote and backward).

Each footnote saves the \Xfnote (without parameter) to the .ref file (if \openref). We can create the mapping from ⟨gfnotenum⟩ to⟨pgfnotenum⟩ in the macro \fn:⟨fnotenum⟩. Each \Xpage macro sets the \lfnotenum to zero.
The \fnote \{\text{⟨text⟩}\} 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 into the Insert group 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\{\text{⟨text⟩}\} are in right margin at odd pages and they are in left margin at even pages. The \_mnote macro saves its position to .ref file as \_Xmnote without parameter. We define \_mn:\{mnotenum\} as \_right or \_left when the .ref file is read. The ifnum \text{0≤0}\#2 trick returns true if \text{⟨pageno⟩} has a numeric type and false if it is a non-numeric type (Roman numeral, for example). We prefer to use \text{⟨pageno⟩}, but only if it has the numeric type. We use \text{⟨gpageno⟩} in other cases.

User can declare \fixmnotes\left or \fixmnotes\right. It defines \_mnotesfixed as \_left or \_right which declares the placement of all marginal notes and such declaration has a precedence.

The \_mnoteD\{\text{⟨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\vskip\-\parskip 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.
The \mnoteA macro does the real work. The \lrmnote{⟨left⟩}{⟨right⟩} uses only first or only second parameter depending on the left or right marginal note.

```
\long \def \mnoteA #1{\incr \mnotenum
\ifx \mnotesfixed \undefined
  \ifcsname _mn:\the \mnotenum \endcsname
  \edef \mnotesfixed{\cs _mn:\the \mnotenum}%
  \else
    \opwarning{unknown \noexpand \mnote side. \TeX me again}\openref
    \incr \unresolvedrefs
    \def \mnotesfixed{\right}\fi
  \fi
\fi
\hbox to0pt{\wref \everypar={%\lrmnote{\kern-\mnotesize \kern-\mnoteindent}{\kern \hsize \kern \mnoteindent}{\kern \mnoteskip \global \mnoteskip=0pt}}%
  \lrmnote{\kern-\mnoteindent}{\kern \hsize \kern \mnoteindent}{\kern \mnoteskip \global \mnoteskip=0pt}}%
}\hss}
```

We don't want to process \note, \fnotemark, \mnote in TOC, headlines nor outlines.

```
\regmacro \fnote{} \fnote{} \fnote{}
\regmacro \fnotemark{} \fnotemark{} \fnotemark{}
\regmacro \mnote{} \mnote{} \mnote{}
```

2.35 Styles

\TeX provides three styles: \report, \letter and \slides. Their behavior is documented in user part of the manual in the section1.7.2 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

We define auxiliary macro first (used by the \address macro)

The \boxlines⟨line-1⟩⟨col⟩⟨line-2⟩⟨col⟩ ... ⟨line-n⟩⟨col⟩ returns to the outer vertical mode a box with ⟨line-1⟩, next box with ⟨line-2⟩ etc. Each box has its natural width. This is reason why we cannot use paragraph mode where each resulting box has the width \hsize. The ⟨col⟩ is set active and \everypar starts \hbox{ and active ⟨col⟩ closes this \hbox by }.

```
\def \boxlines{%\everypar=
  \def \boxlinesE{\ifhmode \egroup \fi}
  \def \nl{\boxlinesE}
  \bgroup \lccode`~=`\^M \lowercase{\egroup \let ~}
  \boxlinesE
  \everypar{\setbox0=\lastbox \endgraf
    \hbox \bgroup \catcode`\^=13 \let \par=\al \aftergroup \boxlinesC}%%
}
```

The \report style initialization macro is defined here.

```
\def \report{
  \tysize[11/13.2]
  \vsize=\dimexpr \topskip + 52 \baselineskip \relax \% added 2020-03-28
  \let \titfont=\chapfont
  \titskip=3ex
  \edef \author#1{\removelastskip \parindent \par \ttindent=\parindent \ttindent}
  \let \author=#1
  \זכ
  \let \parindent=1.2em \iindent=\parindent \ttindent=\parindent
  \footline={\global \footline={\hspace{\rmfixed} \folio \hss}}
}
```

181
The \letter style initialization macro is defined here.

The \letter defines \address and \subject macros.

See the files demo/op-letter-*.tex for usage examples.

The \slides macro reads macro file slides.opm, see the section 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 3mm. If we use 1mm, then the baseline of \footline is 2mm from the bottom page. This is the depth of the \Grey rectangle used for page numbers. It is r-lapped to \hoffset width because left margin = \hoffset = right margin. It is 14mm for narrow pages or 16mm for wide pages.

The \subtit is defined analogically like \tit.
The `\pshow{num}` prints the text in invisible (transparent) font when `\layernum{\langle num\rangle}`. For transparency we need to define special graphics states.

```latex
\def\Transparent {\_transparency255 }
\public \Transparent ;
\def\use#1#2{\_ifnum\layernum#1\relax#2\fi}
\def\pshow#1{\_use{=#1}\Red \_use{<#1}\_Transparent \_ignorespaces}
```

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` 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`.

```latex
\newcount\gilevel
\def\*_{*}{\_relax\_ifmmode\_*\_else\ea\startitem\fi} % defined also in styles.opm
\sdef\_item:X{\_scolor\raise.2ex\fullrectangle{.8ex}\_kern.5em}
\sdef\_item:x{\_scolor\raise.3ex\fullrectangle{.6ex}\_kern.4em}
\style X
\def\egroups{\_par\_global\_gilevel=\_ilevel \_egroup}
everylist={\_novspaces \_ifcase\_ilevel \_or \_style x \_else \_style - \_fi \_addto\egroups{\_egroup}}
```

The default values of `\pg`, i.e. `\pg;`, `\pg+` and `\pg.` are very simple. They are used when `\showsides` is not specified.

```latex
\def\pg#1{\cs{_spg:#1}}
\sdef{_spg:;}{\vfil\break \lfnotenumreset}
\sdef{_spg:.}{\endslides}
\sdef{_spg:+}{\par}
```

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`.

```latex
\def\_endslides{\vfill \_supereject \_byehook \_end}
\def\bye{\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`.

```latex
\def\titfont{\_typosize[42/60]\_bf \_scolor}
\def\subtitfont{\_typosize[20/30]\_bf}
\def\secfont{\_typosize[25/30]\_bf \_scolor}
\nonum \_notoc \_let\_resetnonumnotoc=\_relax
\def\printsec#1{\_par\_abovetitle{\penalty-400}\bigskip
\_secfont \_noindent \_leftskip=0pt plus1fill \_rightskip=\_leftskip
\_printrefnum[@\quad]#1\nbpar}\_insertmark{#1}%
\nobreak \_belowtitle{\_medskip}\_par}
```

When `\slideshow` is active then each page is opened by `\setbox\_slidepage=vbox\group` (roughly speaking) and closed by `\group`. The material is `\un vbox`ed 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 `\layered-text` 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`
\def\\slideopen\ \slideshow\ should be used instead of \slideshow to deactivate it but keep the borders of groups.

\let\slideopen=\relax % first wins
\setbox\slidepage=vbox\bgroup\bgroup

\def\\slideopen\ \slideshowactive
\let\slideopen\relax
\sdef{\spg:;}{\egroups\vfil\break\lfnotenumreset\bgroup\setilevel}
\sdef{\spg:.}{\egroups\endslides}
\sdef{\spg:+}{\egroups\setilevel
\global\layernum=0}
\let\layersopen=\egroup
\let\layersclose=\bgroup

\def\\closepage{
\ifnum\layernum<\maxlayers
\incr\layernum \decr\pageno
\else
\fi}
\ifvoid\slidepage
\unvcopy\slidepage\vfil\break
\else

\begingroup
\hspace{\parindent}
\par
\egroup
\setilevel
\ifnum\layernum<\maxlayers
\incr\layernum \decr\pageno
\else
\fi
\fi

\def\\resetpage{
\global\setbox\slidepage=\box\voidbox
\global\setbox\slidepageB=\box\voidbox
\lfnotenumreset
}

\def\\closepage{
\ifnum\layernum<\maxlayers
\incr\layernum \decr\pageno
\else
\fi
\global\layernum=0
\global\maxlayers=\layernum
\global\slidelines % to prevent hyperlink-dests duplication
}
When \texttt{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{slideshow} creates boxes of typesetting material and copies them to more pages. So, we have to suppress creating destinations in these boxes. This is done in the \_slidelinks macro. We can move creating these destinations to the output routine. \_sdestbox is saved value of the original \_destbox which is redefined to do only \addto\_destboxes\{\_sdestbox[\langle label\rangle]\}. All destinations saved to \_destboxes are created at the start of the next output routine in the \_pagedest macro. The output routine removes \_destboxes, so each destination is created only once.

Limitations of this solution: destinations are only at the start of the page, no at the real place where \w\texttt{label} was used. The first “virtual” page where \w\texttt{label} is used includes its destination. If you want to go to the final page of the partially uncovering ideas then use \texttt{\label[\langle label\rangle]\w\texttt{label}\texttt{text}} in the last part of the page (before \texttt{\pg;}) o use \texttt{\pgref instead \wref}.\texttt{slides.opm}

\begin{verbatim}
\def\slidelinks{%
  \def \_destbox[#1]{\edef \tmp{\noexpand \sdestbox[#1]}%}
  \global \ea \addto \ea \destboxes \ea{\tmp}}%
\def \_pagedest {%
  \hbox{\def \destheight{25pt} \sdestbox[pg:\the\gpageno] \destboxes}%
  \noninterlineskip \gdef \destboxes{}%}
\ifx \dest \destactive \else \let \pagedest=\relax \fi%
}
\let \sdestbox = \destbox
\def \destboxes{} % initial value of \_destboxes
\let \bibgl=\global % \advance \bibnum must be global if they are at more pages
\end{verbatim}

The \_settinglayer is used in the \_layertext macro to prevent printing “Duplicate label” warning when it is expanded. It is done by special value of \_slideshook (used by the \w\texttt{label} macro). Moreover, the warning about illegal use of \w\texttt{bib}, \w\texttt{usebib} in \layers environment is activated.

\texttt{slides.opm}

\begin{verbatim}
\def\settinglayer{%
  \def \slideshook #1#2{}%
  \def \bibB[#1]{\nousebib} \def \usebib/##1 (##2) {\nousebib}%
}
\def \nousebib{\opwarning{Don’t use \noexpand\bib nor \noexpand\usebib in \string\layers}}%
\end{verbatim}

Default \texttt{\layers\ (num)} macro (when \texttt{slideshow} is not activated) is simple. It prints the \texttt{\layered-text} with \texttt{\layernum=\numexpr\num+1\relax} because we need the result after last layer is processed.

\texttt{slides.opm}

\begin{verbatim}
\long\def\layers #1 \#2\endlayers{\par}
\let\layersopen=\string\numexpr\#1+1\relax \#2\endgraf}
\let\layersclose=\relax
\let\fnotenumpages\public \fnotenumpages ;
\end{verbatim}

We must to 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.

\texttt{slides.opm}

\begin{verbatim}
\long\def\fnotenumpages {\def \fnotenum{\the \lfnotenum} \pgfnotefalse
\def \fnotenumreset{\global \lfnotenum=0 }}
\let \lfnotenumreset=\relax
\public \fnotenumpages ;
\end{verbatim}

\subsection{2.36 Logos}

\texttt{logos.opm}

\begin{verbatim}
\codedecl \TeX {Logos TeX, LuaTeX, etc. <2020-02-28>} % preloaded in format
\end{verbatim}

Despite plain \TeX each macro for logos ends by \texttt{\ignoreslash}. This macro ignores the next slash if it is present. You can use \texttt{\TeX/ like this} for protecting the space following the logo. This is visually more comfortable. The macros \texttt{\TeX, \OpTeX, \LuaTeX, \XeTeX} are defined.

185
The \_slantcorr macro expands to the slant-correction of the current font. It is used to shifting A if the \LaTeX logo is in italic.

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 the file uni-lcuc.opm is lefted by \endinput.

If the file UnicodeData.txt does not exists then internal data are used. They follow to the end of the file uni-lcuc.opm.
\_everyeof={;;;;;;;;;} % end of file
\_endgroup \_endinput \_fi % endinput here, if UnicodeData.txt was loaded

% If UnicodeData.txt not found, we have internal copy here from csplain, 2014:
\_def \_tmp #1 #2 {\_ifx^#1^\_else
\_lccode"#1="#1
\_ifx.#2\
\_uccode"#1="#1
\_else
\_uccode"#2="#2
\_lccode"#2="#1
\_uccode"#1="#2
\_fi
\_ea \_tmp \_fi
\_tmp
00AA .
00B5 039C
00BA .
00E0 00C0
00E1 00C1
00E2 00C2
00E3 00C3
00E4 00C4
00E5 00C5
...
\_langw\langle lang-tag\rangle \{chapter\} \{table\} \{figure\} \{subject\} for declaring them.
\today macro needs auto-generated words for each name of the month.
\_month\langle lang-tag\rangle \{January\} \{February\} ... \{December\} is used for decaring them.

The language-dependent format for printing date should be declared like

\_langb\langle lang-tag\rangle \{and\} \{et-al\} \{ed\} \{cit\} \{vol\} \{no\} \{pp\} \{ed\} \{eds\} \\
\{avail-from\} \{avail-to\} \{ba-thesis\} \{ma-thesis\} \{phd-thesis\}. It is used similar way as the
\_langb above. Both these macros are used in lang-data.opm file, see the end of section 2.37.3.

\_langb\langle lang-tag\rangle \{and\} \{et-al\} \{ed\} \{cit\} \{vol\} \{no\} \{pp\} \{ed\} \{eds\} \\
\{avail-from\} \{avail-to\} \{ba-thesis\} \{ma-thesis\} \{phd-thesis\}. It is used similar way as the
\_langb 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.
\_month\langle lang-tag\rangle \{January\} \{February\} ... \{December\} is used for decaring them.

The language-dependent format for printing date should be declared like

2.37.2 Multilingual phrases and quotation marks

Four words are generated by OpTeX macros: “Chapter”, “Table”, “Figure” and “Subject”. These phrases are generated depending on the current value of the \_language register, if you use \_mtext\langle phrase-id\rangle, specially \_mtext\langle chap\rangle, \_mtext\langle t\rangle, \_mtext\langle f\rangle or \_mtext\langle subj\rangle. If your macros generate more words then you can define such words by \sdef\langle phrase-id\rangle :\langle lang-tag\rangle where \langle phrase-id\rangle is a label for the declared word and \langle lang-tag\rangle is a language shortcut declared by \_preplang.

We can declare such language-dependent words by

\_def \_langw #1 #2 #3 #4 #5 {\_sdef\langle mtext:chap\rangle\langle #1\rangle}{\_sdef\langle mtext:t\rangle\langle #2\rangle}{\_sdef\langle mtext:f\rangle\langle #3\rangle}{\_sdef\langle mtext:subj\rangle\langle #4\rangle}{\_sdef\langle mtext:ed\rangle\langle #5\rangle}

More phrases are auto-generated in bibliography references. They are declared by

\_langb\langle lang-tag\rangle \{and\} \{et-al\} \{ed\} \{cit\} \{vol\} \{no\} \{pp\} \{ed\} \{eds\} \\
\{avail-from\} \{avail-to\} \{ba-thesis\} \{ma-thesis\} \{phd-thesis\}. It is used similar way as the
\_langb 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.
\_month\langle lang-tag\rangle \{January\} \{February\} ... \{December\} is used for decaring them.
This example declares date format for English where ⟨lang-tag⟩ is en.

\def\month#1 \#2 \#3 \#4 \#5 \#6 \#7 \% \def\sdef{mt:today:en}{\mtext{m\_the\_month} \_the\_day, \_the\_year}

\begin{verbatim}
\_sdef{\_mt:today:en}{\_mtext{m\_the\_month} \_the\_day, \_the\_year}
\end{verbatim}

Quotes should be tagged by \"⟨text⟩" and \ ⟨text⟩ " if \iso-codequotes is declared at beginning of the document (for example \enquotes). If not, then the control sequences \" and ' are undefined. Remember, that they are used in another meaning when the \oldaccents command is used. The macros \" and ' are not defined as \protected because we need their expansion when \outlines are created. User can declare quotes by \quoteschars, \csquotes, \frquotes, \dequotes, \skquotes are alternative quotes. or use \altquotes to swap between the meaning of these two types of quotes. \enquotes, \csquotes, \frquotes, \dequotes, \skquotes are defined here. Languages in general provide the \quotes declaration macro. It declares the quotation marks depending on the actual selected language. For example, \eslang \quotes declares Spanish language including its quotation marks used for \"⟨text⟩" and \ ⟨text⟩ " . The language-dependent quotation marks should be declared by \quotemacro ⟨lang-tag⟩ \{ ⟨clq⟩ ⟨crq⟩ ⟨clq⟩ ⟨crq⟩ \} in the lang-data.opm file.

The \quoteschars⟨lqq⟩⟨rqq⟩⟨lqq⟩⟨rqq⟩ defines " and ’ as \qqB in normal mode and as expandable macros in outline mode. We want to well process the common cases: \" " and ’ " \cdot " . 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 \(" \cdot "\). This is the reason why the sub-verbatim mode is used when the first character is { in the parameter.

The " is defined as \qqA\qqB⟨lqq⟩⟨rqq⟩ and \’ as \qqA\qqC⟨lqq⟩⟨rqq⟩ . The \qqA\qqB⟨clq⟩⟨crq⟩ runs \qqB⟨lqq⟩⟨rqq⟩⟨text⟩ ,

The \regquotes""{⟨L⟩}{⟨R⟩} does \def\"1"{⟨L⟩\#1⟨R⟩} for outlines but the " separator is active (because " and ’ are active in \pdfunidef).

\begin{verbatim}
\_sdef{\_mt:today:en}{\_mtext{m\_the\_month} \_the\_day, \_the\_year}
\end{verbatim}

Sometimes should be usable to leave the markup "such" or 'such' i.e. without the first backslash. Then you can make the characters " and ’ active by the \activequotes macro and leave quotes without the first backslash. First, declare \iso-codequotes, then \altquotes (if needed) and finally \activequotes.

\begin{verbatim}
\_sdef{\_mt:today:en}{\_mtext{m\_the\_month} \_the\_day, \_the\_year}
\end{verbatim}

188
2.37.3 Languages declaration

\_preplang \{lang-id\} \{LongName\} \{lang-tag\} \{hyph-tag\} \{lr-hyph\} declares a new language. The parameters (separated by space) are:

- \{lang-id\}: language identifier. It should be derived from ISO 639-1 code but additional letters can be eventually added because \{lang-id\} must be used uniquely in the whole declaration list. The \_preplang macro creates the language switch \{lang-id\}lang and defines also \{lang-id\}lang as a macro which expands to \{lang-id\}lang. For example, \_preplang cs Czech ... creates \cslang as the language switch and defines \def\cslang{\_cslang}.

- \{LongName\}: full name of the language.

- \{lang-tag\}: language tag, which is used for setting language-dependent phrases and sorting data. If a language have two or more hyphenation patterns but a single phrases set, then we declare this language more than once with the same \{lang-tag\} but different \{lr-hyph\}.

- \{hyph-tag\}: a part of the file name where the hyphenation patterns are prepared in Unicode. The full file name is hyph-\{hyph-tag\}.tex. If \{hyph-tag\} is {} then no hyphenation patterns are loaded.

- \{lr-hyph\}: two digits, they denote \left\hyphen\min\ and \right\hyphen\min\ values.

\_preplang allocates a new internal number by \newlanguage\{lang-id\}Patt which will be bound to the hyphenation patterns. But the patterns nor other language data are not read at this moment. The \{lang-id\}lang is defined as \_langinit. When the \{lang-id\}lang switch is used firstly in a document then the language is initialized, i.e. hyphenation patterns and language-dependent data are read. The \{lang-id\}lang is re-defined itself after such initialization. \_preplang does also \def\ulan:\{longname\} \{\{lang-id\}\}, this is needed for the \uselanguage macro.

The \_preplang macro adds \{lang-id\} (\{LongName\}) to the \langlist macro which is accessible by \langlist. It can be used for reporting declared languages.

All languages with hyphenation patterns provided by \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.
\_preplangmore \{lang-id\}\{space\}\{\text\} declares more activities of the language switch. The \text is processed whenever \_lang is invoked. If \_preplangmore is not declared for given language

190
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.

```
\def\preplangmore #1 #2{\ea \gdef \csname _langspecific:#1\endcsname{#2}}
\def\langdefault{\frenchspacing}
```

The \_langdefault macro does:
- The internal language \langle number\rangle is extracted from \_the\langle lang-id\rangle Patt.
- \def \lan:(number) \{(lang-tag)\} for mapping from \language number to the \langle lang-tag\rangle.
- loads hyph:\langle hyph-tag\rangle .tex file with hyphenation patterns when \language=\langle number\rangle.
- loads the part of lang-data.opm file with language-dependent phrases using \_langinput.
- \def \langle lang-id\rangle lang \{\_uselang{\langle lang-id\rangle }\_\langle lang-id\rangle Patt \langle pre-hyph\rangle\langle post-hyph\rangle\}. i.e. the switch redefines itself for doing a “normal job” when the language switch is used repeatedly.
- Runs itself (i.e. \_\langle lang-id\rangle lang) again for doing the “normal job” firstly.

```
\def\langinit #1 #2 #3 #4 #5 #6 #7{% \switch lang-id(LongName)lang-tag[hyph-file]lr-hyph
\sxdef{_lan:the\csname #2Patt\endcsname}{#4}%
\begingroup \setbox0=\vbox{% we don’t want spaces in horizontal mode
\setctable\optexcatcodes \language=\cs{\#2Patt} \relax
\ifx^#5^\else
\wlog{Loading hyphenation for #3: \string\language=\space(#5)}%
\let\patterns=\patterns \let\hyphenation=\hyphenation \def\message##1{}%
\isfile{hyph-#5}\iftrue \input{hyph-#5}\fi
\else \opwarning{No hyph. patterns #5 for #3, missing package?}\fi
\fi
% loading language data:
\_langinput(#4)%
\endgroup
\xdef#1{\noexpand\_uselang{#2}\_csname \#2Patt\endcsname #6#7}%
\_uselang{\langle lang-id\rangle }\_\langle lang-id\rangle Patt \langle pre-hyph\rangle\langle post-hyph\rangle is used as “normal job” of the switch. It sets \language, \lefthyphenmin, \righthyphenmin. Finally, it runs data from \_preplangmore or runs \_langdefault.
The \uselanguage \{LongName\} macro is defined here (for compatibility with e-plain users). Its parameter is case insensitive.

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 lang-data.opm file. Appropriate parts of this file are read by \langinput\{\lang-tag\}. First few lines of the file looks like:

```latex
\codedecl \langdata {Language dependent data <2022-06-28>} % 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.}
\monthw en January February March April May June July August September October November December
\sdef{\mt:today:en}{\mtext{m\_the\_month} \_the\_day, \_the\_year} % date format
\quotationmarks en {"''}

%_let \_sortingdataen = \_sortingdatalatin % set already, see section 2.33, makeindex.opm
%_let \_ignoredcharsen = \_ignoredcharsgeneric
%_def \_compoundcharsen {}

\langdata cs {Czech} % --------------------------------------------------------------
\langw cs Kapitola Tabulka Obrázek Věc
\langb cs {, and } { et al.} { ed.} {cit.} {Vol.} {No.} {pp.} {p.} {,ed.} {,eds.}
\monthw cs ledna února března dubna května červen
\sdef{\mt:today:cs}{\the\_day. \mtext{m\_the\_month} \_the\_year} % date format
\quotationmarks cs {"''}

%_let \_sortingdatacs = \_sortingdatalatin % see \_compoundchars in section 2.33
%_let \_ignoredcharscs = \_ignoredcharsgeneric
%_def \_compoundcharscs {ch:^^T Ch:^^U CH:^^V} % see \_compoundchars in section 2.33
%_def \_compoundcharsde {ß:ss} % alternative sorting

%_def \_compoundcharsde {ß:ss Ä:AE Ö:OE Ü:UE ä:ae ö:oe ü:ue} % alternative sorting

\langdata de {German} % --------------------------------------------------------------
\langw de Kapitel Tabelle Abbildung Betreff
\quotationmarks de {"",’}
```

There are analogical declaration for more languages here. Unfortunately, this file is far for completeness. I welcome you send me a part of declaration for your language.
If your language is missing in this file then a warning is reported during language initialization. You can create your private declaration in your macros (analogical as in the `lang-data.opm` file but without the \_langdata prefix). Then you will want to remove the warning about missing data. This can be done by `\nolanginput{⟨lang-tag⟩}` given before initialization of your language.

The whole file `lang-data.opm` is not preloaded in the format because I suppose a plenty languages here and I don’t want to waste the \TeX memory by these declarations. Each part of this file prefixed by \_langdata{⟨lang-tag⟩} {⟨LongName⟩} is read separately when `\langinput{⟨lang-tag⟩}` is used. And it is used in the \langinit macro (i.e. when the language is initialized), so the appropriate part of this file is read automatically on demand.

If the part of the `lang-data.opm` concerned by ⟨lang-tag⟩ is read already then \li{⟨lang-tag⟩} is set to R and we don’t read this part of the file again.

Data of two preferred languages are preloaded in the format:

\langinput{en} \langinput{cs}

### 2.38 Other macros

Miscellaneous macros are here.

\useOpTeX and \useoptex are declared as `\relax`.

The `\lastpage` and `\totalpages` get the information from the `\currpage`. The `\Xpage` from .ref file sets the `\currpage`.

We need `\uv`, `\crq`, `\fqq`, `\fqqq`, `\uslang`, `\hyph`, `\chyph`, `\shyph`, for backward compatibility with \CSCplain. Codes are set according to Unicode because we are using Czech only in Unicode when Lua\TeX is used.

```latex
% for compatibility with csplain:
\chardef\clqq=8222 \chardef\frqq=8220
\chardef\fqq=171 \chardef\frqq=187
\chardef\promile=8240
\def\uv#1{\clqq#1\crqq}
```

```latex
\let\uslang=\enlang \let\hyph=\enlang
\let\chyph=\enlang \let\shyph=\sklang
\let\csUnicode=\csPatt \let\czUnicode=\csPatt \let\skUnicode=\skPatt
```
The \letfont was used in \cscplain instead of \fontlet.

\let \letfont = \_fontlet

Non-breaking space in Unicode.

\let ^^a0=~

Old macro packages need these funny control sequences. We don’t use them in new macros.

We don’t want to read opmac.tex unless \input opmac is specified.

We allow empty lines in math formulae. It is more comfortable.

Lorem ipsum can be printed by \lipsum\langle range\rangle or \lorem\langle range\rangle, for example \lipsum[3] or \lipsum[112-121], max=150.

First usage of \lipsum reads the \TeX 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\langle number\rangle or \lipsum\langle number\rangle.1 (i.e. dot after the parameter). The first case prints the paragraph \langle number\rangle without the final \par and the second case prints only first sentence from the paragraph \langle number\rangle using \lipsumdot.

\lipsum adds \par after each printed paragraph. If you don’t need such \par here, use \lipsumtext\langle number\rangle or \lipsum\langle number\rangle.1 (i.e. dot after the parameter). The first case prints the paragraph \langle number\rangle without the final \par and the second case prints only first sentence from the paragraph \langle number\rangle using \lipsumdot.
LuaTeX version 1.14 and newer provides \partokenname which allows to specify something different than \par at empty lines. We set \_par (see below) in OpTeX version 1.04+ and newer. Some macros were rewritten due to this change. And we copy old versions of these changed macros here in order to allow to use older LuaTeX versions where \partokenname is not provided.

Note that your macros where a parameter is separated by the empty line must be changed too. Use \def\macro #1\_par{...} instead \def\macro #1\par{...}.

2.39 Lua code embedded to the format

The file optex.lua is loaded into the format in optex.ini as byte-code and initialized by \everyjob, see section 2.1.
The file implements part of the functionality from `luatexbase` namespace, nowadays defined by \LaTeX kernel. `luatexbase` deals with modules, allocators, and callback management. Callback management is a nice extension and is actually used in \TeX. Other functions are defined more or less just to suit luatexload’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.

```
local fmt = string.format

2.39.1 General

Define namespace where some \TeX functions will be added.

optex = optex or {}

Error function used by following functions for critical errors.

local function err(message)
  error("error: '..message..'\n")
end

For a \chardef'd, \countdef'd, etc., csname return corresponding register number. The responsibility of providing a \XXdef'd name is on the caller.

function registernumber(name)
  return token.create(name).index
end

MD5 hash of given file.

function mdfive(file)
  local fh = io.open(file, "rb")
  if fh then
    local data = fh:read("*a")
    fh:close()
    tex.print(md5.sumhexa(data))
  end
end

2.39.2 Allocators

alloc = alloc or {}

An attribute allocator in Lua that cooperates with normal \TeX allocator.

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.

local function_table = lua.get_functions_table()

function define_lua_command(csname, fn, ...)
  local luafnalloc = #function_table + 1
  token.set_lua(csname, luafnalloc, ...) -- WARNING: needs \TeX 1.08 (2019) or newer
  function_table[luafnalloc] = fn
end
```
provides_module is needed by older version of luaotfload

2.39.3 Callbacks

callback = callback or ()

Save callback.register function for internal use.

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.

local callback_functions = {}

Table that maps callback name to a list of descriptions of its added functions. The order corresponds with callback_functions.

local callback_description = {}

Table used to differentiate user callbacks from standard callbacks. Contains user callbacks as keys.

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).

local default_functions = {}

Table that maps standard (and later user) callback names to their types.

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_font_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
}
Return a list containing descriptions of added callback functions for specific callback.

Create a user callback that can only be called manually using call_callback. A default function is only needed by "exclusive" callbacks.
elseif not valid_callback_types[cbtype] then
  err("cannot create callback '"..name.. "' with invalid callback type '"..cbtype.."'")
  elseif type == "exclusive" and not default then
    err("unable to create exclusive callback '"..name.."', default function is required")
  end

callback_types[name] = cbtype
default_functions[name] = default or nil
user_callbacks[name] = true
end

Add a function to the list of functions executed when callback is called. For standard luatex callback a proxy function that calls our machinery is registered as the real callback function. This doesn’t happen for user callbacks, that are called manually by user using call_callback or for standard callbacks that have default functions – like mlist_to_hlist (see below).

local call_callback
function callback.add_to_callback(name, fn, description)
  if user_callbacks[name] or callback_functions[name] or default_functions[name] then
    -- either:
    -- a) user callback - no need to register anything
    -- b) standard callback that has already been registered
    -- c) standard callback with default function registered separately
    -- (mlist_to_hlist)
  elseif callback_types[name] then
    -- This is a standard luatex callback with first function being added,
    -- register a proxy function as a real callback. Assert, so we know
    -- when things break, like when callbacks get redefined by future
    -- luatex.
    callback_register(name, function(...)
      return call_callback(name, ...)
  end
  else
    err("cannot add to callback '"..name.."' - no such callback exists")
  end

  -- add function to callback list for this callback
  callback_functions[name] = callback_functions[name] or {}
  table.insert(callback_functions[name], fn)

  -- add description to description list
  callback_description[name] = callback_description[name] or {}
  table.insert(callback_description[name], description)
end

Remove a function from the list of functions executed when callback is called. If last function in the list is removed delete the list entirely.

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.
end
local function reverse_ipairs(t)
    local i, n = #t + 1, 1
    return function()
        i = i - 1
        if i >= n then
            return i, t[i]
        end
    end
end

Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

local cbtype = callback_types[name]
-- either take added functions or the default function if there is one
local functions = callback_functions[name] or {default_functions[name]}

if cbtype == nil then
    err("cannot call callback ".name." - no such callback exists")
elseif cbtype == "exclusive" then
    -- only one function, atleast default function is guaranteed by create_callback
    return functions[1]()
elseif cbtype == "simple" then
    -- call all functions one after another, no passing of data
    for _, fn in ipairs(functions) do
        fn()
    end
    return
elseif cbtype == "data" then
    -- pass data (first argument) from one function to other, while keeping other arguments
    local data = (...)
    for _, fn in ipairs(functions) do
        data = fn(data, select(2, ...))
    end
    return data
end

-- list and reverselist are like data, but "true" keeps data (head node)
-- unchanged and "false" ends the chain immediately
local iter
if cbtype == "list" then
    iter = ipairs
else 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
        local changed = true
        head = new_head
    end
end
if changed then
    return new_head
end
return false
head = new_head
changed = true
end
end
return not changed or head
call_callback = callback.call_callback

Create “virtual” callbacks pre/post_mlist_to_hlist_filter by setting mlist_to_hlist callback. The default behaviour of mlist_to_hlist is kept by using a default function, but it can still be overridden by using add_to_callback.

default_functions["mlist_to_hlist"] = node.mlist_to_hlist
callback.create_callback("pre_mlist_to_hlist_filter", "list")
callback.create_callback("post_mlist_to_hlist_filter", "reverselist")
callback_register("mlist_to_hlist", function(head, ...)
-- pre_mlist_to_hlist_filter
local new_head = call_callback("pre_mlist_to_hlist_filter", head, ...)
if new_head == false then
node.flush_list(head)
return nil
elseif new_head ~= true then
head = new_head
end
-- mlist_to_hlist means either added functions or standard luatex behavior
-- of node.mlist_to_hlist (handled by default function)
head = call_callback("mlist_to_hlist", head, ...)
-- post_mlist_to_hlist_filter
new_head = call_callback("post_mlist_to_hlist_filter", head, ...)
if new_head == false then
node.flush_list(head)
return nil
elseif new_head ~= true then
head = new_head
end
return head
end)

For preprocessing boxes just before shipout we define custom callback. This is used for coloring based on attributes. There is however a challenge - how to call this callback? We could redefine \shipout and \pdfxform (which both run ship_out procedure internally), but they would lose their primitive meaning – i.e. \immediate wouldn’t work with \pdfxform. The compromise is to require anyone to run \_preshipout{destination box number}{box specification} just before \shipout or \pdfxform if they want to call pre_shipout_filter (and achieve colors and possibly more).

callback.create_callback("pre_shipout_filter", "list")

local tex_setbox = tex.setbox
local token_scanint = token.scan_int
local token_scanlist = token.scan_list
define_lua_command("_preshipout", function()
local boxnum = token_scanint()
local head = token_scanlist()
head = call_callback("pre_shipout_filter", head)
tex_setbox(boxnum, head)
end)

Compatibility with \LaTeX through luatexbase namespace. Needed for luaotfload.

luatexbase = {
  registernumber = registernumber,
  attributes = attributes,
  provides_module = provides_module,
  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,
}
\begin{verbatim}
callbacktypes = {}

\tracingmacros callback registered. Use \tracingmacros=3 or \tracingmacros=4 if you want to see the result.

\begin{verbatim}
callback.add_to_callback("input_level_string", function(n)
    if tex.tracingmacros > 3 then
        return ": n \n"
    elseif tex.tracingmacros > 2 then
        return ": . string_rep("\n",n)
    else
        return ":
    end
end

\end{verbatim}
\end{verbatim}

401 callbacktypes = {}
402 }

2.39.4 Management of PDF page resources
Traditionally, pdf\TeX allowed managing PDF page resources (graphics states, patterns, shadings, etc.) using a single toks register, \pdfpageresources. This is insufficient due to the expected PDF object structure and also because many “packages” want to add page resources and thus fight for the access to that register. We add a finer alternative, which allows adding different kinds of resources to a global page resources dictionary. Note that some resource types (fonts and XObjects) are already managed by Lua\TeX and shouldn’t be added!

XObject forms can also use resources, but there are several ways to make Lua\TeX reference resources from forms. It is hence left up to the user to insert page resources managed by us, if they need them. For that, use pdf.get_page_resources(), or the below \TeX alternative for that.

\begin{verbatim}
local pdfdict_mt = {
    __tostring = function(dict)
        local out = {"<"}
        for k, v in pairs(dict) do
            out[#out+1] = fmt("/\%s %s", tostring(k), tostring(v))
        end
        out[#out+1] = ">")
        return table.concat(out, "\n")
    end,
}
local function pdf_dict(t)
    return setmetatable(t or {}, pdfdict_mt)
end

local resource_dict_objects = {}
local page_resources = {}

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 /\%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
    return pdf.get_page_resources() or ""
end

function pdf.get_page_resources()
    return pdf.getpageresources() or ""
end

New “pseudo” primitives are introduced. \_addpageresource{(type)}{(PDF name)}{(PDF dict)} adds more resources of given resource (type) to our data structure. \_pageresources expands to the saved (type)s and object numbers.

\begin{verbatim}
define_lua_command("_addpageresource", function()
    pdf.add_page_resource(token.scan_string(), token.scan_string(), token.scan_string())
end)
define_lua_command("_pageresources", function()
tex.print(pdf.get_page_resources())
end)
\end{verbatim}

202
We write the objects with resources to the PDF file in the `finish_pddfie` callback.

```latex
475 callback.add_to_callback("finish_pddfie", function()
476 for type, dict in pairs(page_resources) do
477 local obj = resource_dict_objects[type]
478 pdf.immediateobj(obj, tostring(dict))
479 end
480 end)
```

### 2.39.5 Handling of colors and transparency using attributes

Because LuaTeX doesn't do anything with attributes, we have to add meaning to them. We do this by intercepting TeX just before it ships out a page and inject PDF literals according to attributes.

```latex
488 local node_id = node.id
489 local node_subtype = node.subtype
490 local glyph_id = node_id("glyph")
491 local rule_id = node_id("rule")
492 local glue_id = node_id("glue")
493 local hlist_id = node_id("hlist")
494 local vlist_id = node_id("vlist")
495 local disc_id = node_id("disc")
496 local whatsit_id = node_id("whatsit")
497 local pdfliteral_id = node_subtype("pdf_literal")
498 local pdfsave_id = node_subtype("pdf_save")
499 local pdfrestore_id = node_subtype("pdf_restore")
500 local token_getmacro = token.get_macro
501
502 local direct = node.direct
503 local todirect = direct.todirect
504 local tonode = direct.tonode
505 local getfield = direct.getfield
506 local setfield = direct.setfield
507 local getwhd = direct.getwhd
508 local getid = direct.getid
509 local getlist = direct.getlist
510 local setlist = direct.setlist
511 local getleader = direct.getleader
512 local getattribute = direct.get_attribute
513 local insertbefore = direct.insert_before
514 local copy = direct.copy
515 local traverse = direct.traverse
516 local one_bp = tex.sp("1bp")
```

The attribute for coloring is allocated in `colors.opm`

```latex
519 local color_attribute = registernumber("_colorattr")
520 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.

```latex
525 local pdf_base_literal = direct.new("whatsit", "pdf_literal")
526 setfield(pdf_base_literal, "mode", 2) -- direct mode
527 local function pdfliteral(str)
528 local literal = copy(pdf_base_literal)
529 setfield(literal, "data", str)
530 return literal
531 end
532 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 transparency 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 literalWhatsits just instruct the `\shipout` related procedures to emit the literal.

203
We also set the stroke and non-stroke colors separately. This is because stroke color is not always needed – LuaTeX itself only uses it for rules whose one dimension is less than or equal to 1 bp and for fonts whose mode is set to 1 (outline) or 2 (outline and fill). Catching these cases is a little bit involved. For example rules are problematic, because at this point their dimensions can still be running \((-2^{30})\) – they may or may not be below the one big point limit. Also the text direction is involved. Because of the negative value for running dimensions the simplistic check, while not fully correct, should produce the right results. We currently don’t check for the font mode at all.

Leaders (represented by glue nodes with leader field) are not handled fully. They are problematic, because their content is repeated more times and it would have to be ensured that the coloring would be right even for e.g. leaders that start and end on a different color. We came to conclusion that this is not worth, hence leaders are handled just opaquely and only the attribute of the glue node itself is checked. For setting different colors inside leaders, raw PDF literals have to be used.

We use the node.direct way of working with nodes. This is less safe, and certainly not idiomatic Lua, but faster and codewise more close to the way \(\TeX\) works with nodes.

```lua
local function is_color_needed(head, n, id, subtype) -- returns fill, stroke color needed
    if id == glyph_id then
        return true, false
    elseif id == glue_id then
        n = getleader(n)
        if n then
            return true, true
        end
        return true, false
    elseif id == rule_id then
        local width, height, depth = getwhd(n)
        if width <= one_bp or height + depth <= one_bp then
            -- running \((-2^{30})\) may need both
            return true, true
        end
        return true, false
    elseif id == whatsit_id and (subtype == pdfliteral_id
        or subtype == pdfsave_id
        or subtype == pdfrestore_id) then
        return true, true
    end
    return false, false
end

local function colorize(head, current, current_stroke, current_tr)
    for n, id, subtype in traverse(head) do
        if id == hlist_id or id == vlist_id then
            -- nested list, just recurse
            local list = getlist(n)
            list, current, current_stroke, current_tr = colorize(list, current, current_stroke, current_tr)
            setlist(n, list)
        elseif id == disc_id then
            -- at this point only no-break (replace) list is of any interest
            local replace = getfield(n, "replace")
            if replace then
                replace, current, current_stroke, current_tr = colorize(replace, current, current_stroke, current_tr)
                setfield(n, "replace", replace)
            end
        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 newliteral then
                    stroke_color = token_getmacro("_color-s:"..current)
                    newliteral = token_getmacro("_color:"..new)
                    current = new
                end
            end
        end
    end
end
```

204
newliteral = fmt("%s %s", newliteral, stroke_color)

else
    newliteral = stroke_color
end

end

if newtr ~= current_tr and fill_needed then -- (fill_ or stroke_needed) = fill_neded
    if newliteral ~= nil then
        newliteral = fmt("%s /tr%d gs", newliteral, newtr)
    else
        newliteral = fmt("/tr%d gs", newtr)
    end

    current_tr = newtr
end

if newliteral then
    head = insertbefore(head, n, pdfliteral(newliteral))
end

end

return head, current, current_stroke, current_tr

Colorization should be run just before shipout. We use our custom callback for this. See the definition of pre_shipout_filter for details on limitations.

callback.add_to_callback("pre_shipout_filter", function(list)
    -- By setting initial color to -1 we force initial setting of color on
    -- every page. This is useful for transparently supporting other default
    -- colors than black (although it has a price for each normal document).
    local list = colorize(todirect(list), -1, -1, 0)

    return tonode(list)
end, "_colors")

We also hook into luaotfload's handling of color. Instead of the default behavior (inserting colorstack whatsits) we set our own attribute. The hook has to be registered after luaotfload is loaded.

function optex_hook_into_luaotfload()
    if not luaotfload.set_colorhandler then
        return -- old luaotfload, colored fonts will be broken
    end

    local setattribute = direct.set_attribute
    local token_setmacro = token.set_macro
    local color_count = registernumber("_colorcnt")
    local tex_getcount, tex_setcount = tex.getcount, tex.setcount

    luaotfload.set_colorhandler(function(head, n, rgbcolor) -- rgbcolor = "1 0 0 rg"
        local attr = tonumber(token_getmacro("_color::"..rgbcolor))
        if not attr then
            attr = tex_getcount(color_count)
            tex_setcount(color_count, attr + 1)
            local strattr = tostring(attr)
            token_setmacro("_color::"..rgbcolor, strattr)
            token_setmacro("_color:"..strattr, rgbcolor)
            -- no stroke color set

            setattribute(n, color_attribute, attr)
        return head, n
    end)

    -- History:
    -- 2022-03-07 transparency in the colorize() function, current_tr added
    -- 2022-03-05 resources management added
    -- 2021-07-16 support for colors via attributes added
    -- 2020-11-11 optex.lua released
2.40 Printing documentation

The \printdoc \{filename\}\{space\} and \prindoctail \{filename\}\{space\} commands are defined after the file doc.opm is load by \load \{doc\}.

The \printdoc starts reading of given \{filename\} from the second line. The file is read in the listing mode. The \prindoctail 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 \end\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 \{cname\} and \el\{cname\} are defined. The second macro should create a hyperlink using \tmpa where the link name of the \cname is saved and \tmpb where the name of the \cname 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\cname\} lines from optex-doc.ref file. Then we can use following macros:

```
\def\Xindex#12\{\def\#1\{\optexdoclink\}
\optexdoclink%
\def\optexdoclink\%
\def\exlink{url:\optexdocurl\csstring\#cs:\tmpa}\%
\ea_urlactive\ea\{\exlink\}{\Cyan}\{\csstring\tmpb\}
\def\optexdocurl{http://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf}
\isfile[optex-doc.eref]\{\iftrue \input[optex-doc.eref]\fi
```

All \el\{cname\}, where \cname is from optex-doc.ref, have the same meaning: \optexdoclink in this example. And \optexdoclink creates the external link in \Cyan color.
2.40.1 Implementation

General declarations.

Maybe, somebody needs seccc or secccc?

\enddocument can be redefined.

A full page of listing causes underfull \vbox in output routine. We need to add a small tolerance.

The listing mode is implemented here. The \maxlines is maximal lines of code printed in the listing mode. The _catcodedot sets dot as letter in listings (for package documentation where \.foo sequences exist).

The scanner of the control sequences in the listing mode replaces all occurrences of \ by \makecs. This macro reads next tokens and accumulates them to \tmpa as long as they have category 11. It means that \tmpa includes the name of the following control sequence when \makecsF is run. The printing
form of the control sequence is set to \_tmpb and the test of existence \_csname is performed. If it is true then active hyperlink is created. If not, then the first _ or . is removed from \_tmpa and the test is repeated.

```
\def\makecs{\def\_tmpa{\futurelet\next\makecsA}}
\def\makecsA{\ifcat a\relax\next\ea\makecsB\else\ea\makecsF\fi}
\def\makecsB#1{\addto\_tmpa{#1}\futurelet\next\makecsA}
\def\makecsF{\let\_tmpb=\_tmpa\ifx\_tmpa\empty\csstring\%\else\ifcsname ,\_tmpa\endcsname\trycs{el:\_tmpa}{\intlink}\else\remfirstunderscoreordot\_tmpa\ifx\_tmpa\empty\let\_tmpa=\_tmpb\fi\fi\fi\fi\fi}
\def\processinput{%\let\start=\relax\ea\replstring\ea{\_aspace^^J}{^^J}\addto\_tmpb{\_end}\isinlist\_tmpb{\_start^^J}\iftrue\advance\ttline by1\fi\replstring\_tmpb{\_start^^J}{\_start}\replstring\_tmpb{\_start}{}\replstring\_tmpb{^^J\_end}{\_end}\replstring\_tmpb{^^J\_end}{}\replstring\_tmpb{\_end}{}\ea\prepareverbdata\ea\_tmpb\ea{\_tmpb^^J}\replthis{\csstring\%}{\noexpand\makecs}\ea\printverb\_tmpb\end\par\ismextchar\par{\elimindent}%}
\def\remfirstunderscoreordot#1{\ea\remfirstuordotA#1\relax#1}
\def\remfirstuordotA#1#2\relax#3{\if _#1\def#3{#2}\fi\if\string#1.\def#3{#2}\fi}
```

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.

```
\def\intlink{\link[cs:\_tmpa]{\ulinkcolor}{\csstring\_tmpb}}
```

The lines in the listing mode have a yellow background.

```
\def\printcodeline#1{%\_advance\maxlines by-1\_ifnum\maxlines<0\ea\endverbprinting\fi\_ifx\printfilename\relax\_penalty\ttpenalty\fi\vskip-4pt\noindent\rlap{\bgverbcolor\vrule height8pt\hskip-2pt\depth5pt\hsize\vrule width\hsize}\printfilename\indent\printverblinenum #1\par}
\def\printfilename{\hbox to0pt{\hskip\hsize\vbox to0pt{\vss\llap{\fnamecolor\docfile}\kern7.5pt}}\hss}
\let\printfilename=\relax
\everytt={\let\printverblinenum=\relax}
\long\def\endverbprinting#1\end#2\end{\fi\_global\maxlines=100000\noindent\typsize[8/]\dots etc. (see {\tt\fnamecolor\docfile})}
```

\docfile is currently documented file. \printdoc and \printdoctail macros are defined here.

```
\def\docfile{}
\def\printdoc #1 {
\_par\_def\docfile{#1}\_everytt={\ttshift=-15pt\_let\printverblinenum=\_relax}\ea\cod\input #1\_everytt={\let\printverblinenum=\_relax}\def\docfile{}}
```

208
You can do \verb!\vitt{filename} ( {from} - {to} ) {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 \afterload macro.

The \verb!something! will be print as ⟨something⟩.
The \texttt{fw} macro for forward links to user documentation point (given later) is defined here.
Index

There are all control sequences used in OpTEX except \TeX primitives. If you want to know something about \TeX primitives then you can use another index from \TeX in a Nutshell.

\_aboveviskip 130
\_abovedisplayskip 125, 127
\_activequotes 188
\_addcitelist 154–155
\_addcolor 113
\_addextgstate 111, 144
\_additcorr 104
\_addpageregion 144, 202
\_address 25, 181–182
\_addtabitemx 149
\_addtoc 28, 38, 105
\_addtopanelist 76
\_addumathfont 91–92, 94
\_adef 17, 28, 38
\_adots 86
\_advancepageno 105, 107
\_afterfi 28, 41
\_afteritcorr 104
\_afterload 52
\_allocat 40
\_allowbreak 56
\_altquotes 188
\_ascisortingtrue 176
\_atethe 131
\_authorname 158
\_b 58
\_backgroundbox 106
\_backgroundpic 141
\_balancecolumns 152
\_bbchar 80, 96
\_begblock 14, 27, 131
\_begitemss 13–14, 27, 48, 130–131
\_begmulti 19, 27, 48, 152
\_begoutput 105–106, 122
\_begtt 16–18, 27, 47–48, 106, 133, 135
\_begtt 133
\_belonm 130
\_belonmt 125, 127
\_betweencolumns 152
\_bf 8–9, 64–65, 74, 80, 96
\_bgro 37
\_bi 8–9, 64–65, 74, 80, 96
\_bib 20–21, 27, 156
\_bibA 156
\_bibB 156
\_bibg 156
\_bibmark 154, 156, 158
\_bibnn 154
\_bibnum 118, 154
\_biboptions 49, 162
\_bibp 154
\_bibpart 21, 49, 154
\_bibpart 157
\_bibtexhook 49, 157
\_bibwarning 158, 161
\_big 85
\_Big 85
\_bigbreak 56
\_bigg 85
\_Bigg 85
\_biggl 85
\_Biggl 85
\_biggm 85
\_Biggm 85
\_biggr 85
\_Biggr 85
\_bigi 85
\_Bigi 85
\_bign 85
\_Bign 85
\_bigr 85
\_Bigr 85
\_bigs 85
\_Bigs 85
\_bigskip 56
\_bigskipamount 46
\_black 110
\_Blue 21, 110
\_bmod 88
\_boldify 67, 104
\_boldmath 9, 80, 82, 91–94, 103
\_boldpai 93
\_bordermatrix 89
\_bordermatrixwithdelims 89
\_box 181
\_bp 28, 54
\_bp 54
\_bpinta 158, 161
\_bpintb 158, 161
\_bpintc 158, 161
\_bprintv 158, 161
\_bracedparam 52
\_break 56
\_Brown 110
\_bslash 38
\_buildrel 88
\_bye 39, 59
\_byehook 39, 115
\_c 58
\_cal 80, 96
\_caption 10–12, 27, 128
\_captionsep 129
\_cases 89
\_catalogexclude 78
\_catalogmathsample 78
\_catalogonly 78
\_catalogsample 78
\_catcode 53
\_catcodedot 207
\_cdots 86
\_centerline 57
\_chap 10, 12, 17–18, 27, 53, 125, 127
\_chapfont 67, 125
\_chapx 126
\_checkextension 34
\_chyp 24, 193
\_circle 142
\_circles 50
\_cite 12, 20–21, 27, 154, 157
\_citeA 154
\_citeborder 12, 118
\_cite1 155
\_clipincircle 24, 144
\_clipinvel 24, 144
\_clipinpath 144
\_clqq 193
\_cmkcolordef 113
\_cmkgtorgb 112–113
\_cnvinfo 51
\_coc 142
\_cod 28, 33–34, 54
\_code 16–17, 27, 47, 132
\_codeden 28, 33–35
\_colnum 149
\_colorattr 109, 111
\_colorcnt 111
\_colorcrop 112
\_colordef 21–22, 28, 109–112, 114
\_colordefin 112
\_colorprefix 111
\_colsep 48, 152
\_commentchars 18, 133, 135–136
\_commoncolordef 113
\_completer 105–106
\_compoundchars 173–174
\_cong 88
\_corrm 81, 93
\_cramped 91
\_createcolumns 153
\_crl 15, 147, 150
\_crl 15, 145, 148, 150
\_crl 15, 150
\_crl 15, 145, 150
\_crlp 15, 145, 150
211