OpTEx

Format Based on Plain TEx and OPmac

Petr Olšák, 2020, 2021

http://petr.olsak.net/optex

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

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

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

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

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

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

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

1 User documentation

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

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

2.1 The main initialization file ................................. 30
2.2 Concept of namespaces of control sequences ............... 32
   2.2.1 Prefixing internal control sequences .................. 32
   2.2.2 Namespace of control sequences for users .............. 32
   2.2.3 Macro files syntax ....................................... 33
   2.2.4 Name spaces for package writers ....................... 33
   2.2.5 Summary about rules for external macro files published for OpTeX ............................ 33
   2.2.6 The implementation of the namespaces .................. 34
Chapter 1
User documentation

1.1 Starting with OpTEX

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

    optex document

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

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

A minimal document should be

\fontfam[LMfonts]
Hello World! \bye

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

A somewhat larger example with common settings should be:

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

Tady je zkušební textík v českém jazyce.
\bye

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

1.2 Page layout

1.2.1 Setting the margins

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

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

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

Parameters are:

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

\textsuperscript{1} This is a technical limitation of Lua\TeX\ 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 \texttt{\hsize} is set. Else \texttt{\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 (\texttt{\vsize} instead \texttt{\hsize} is used). Examples:

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

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

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

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

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

\begin{verbatim}
\margins/1 (100,200) (7,7,7,7)mm
\end{verbatim}

declares the paper 100×200 mm with all four margins 7 mm. The spaces before and after \langle fmt \rangle parameter are necessary.

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

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

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

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

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

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

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

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

### 1.2.3 Footnotes and marginal notes

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

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

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

The marginal note can be printed by the \texttt{\textbackslash mnote\{\text\}} macro. The \texttt{\textbackslash 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{\textbackslash 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{\textbackslash 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{\textbackslash 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\{\textbackslash Family\-name\}}. The argument \texttt{\textbackslash 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 OpTeX are listed on the terminal and in the log file. If you write \texttt{\fontfam[catalog]} then a catalog of all fonts registered in

\textsuperscript{2} You can declare \texttt{\textbackslash fnotenumglobal} if you want footnotes numbered in whole document from one or \texttt{\textbackslash fnotenumpages} if you want footnotes numbered at each page from one. Default setting is \texttt{\textbackslash fnotenumchapters}
OpTEX and available in your TeX system is printed. The instructions on how to register your own font family are appended in the catalog.

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

Example:

\begin{verbatim}
This is \em important text.  \%= This is \it important\/> text.
\it This is \em important text. \%= This is/ \rm important text.
\bf This is \em important text. \%= This is \bi important\/> text.
\bi This is \em important text. \%= This is/ \bf important text.
\end{verbatim}

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

1.3.2 Font sizes

The command \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:

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

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

\begin{verbatim}
\typoscale[800/800] \%= fonts and baselineskip re-size to 80 \%
\typoscale[1440/1440] \%= fonts bigger 1.44times (\magstep2 expands to 1440)
\end{verbatim}

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 \mainfsize and \mainbaselineskip.
The `\typoscale` command does scaling with respect to current values by default. If you want to do it with respect to the main values, type `\scalemain` immediately before `\typoscale` command.

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

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

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

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

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

1.3.3 Typesetting math

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

\LaTeX{} 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{\boldmath}` variant. Most declared text font families (see `\texttt{\fontfam}` in the section 1.3.1) are configured with a recommended Unicode math font. This font is automatically loaded unless you specify `\texttt{\noloadmath}` before first `\texttt{\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{\loadmath\{\langle font-file\rangle\}}` or `\texttt{\loadmath\{\langle font-name\rangle\}}` before first `\texttt{\fontfam}` command.

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

The following math alphabets are available:

\texttt{\mit} % mathematical variables \hspace{1cm} abc\text{-}xyz, ABC\text{-}XYZ
\texttt{\it} % text italics \hspace{1cm} abc\text{-}xyz, ABC\text{-}XYZ
\texttt{\rm} % text roman \hspace{1cm} abc\text{-}xyz, ABC\text{-}XYZ
\texttt{\cal} % normal calligraphics \hspace{1cm} ABC\text{-}XYZ
\texttt{\script} % script \hspace{1cm} \mathcal{ABC}\text{-}XYZ
\texttt{\frak} % fracture \hspace{1cm} abc\text{-}xyz, ABC\text{-}XYZ
\texttt{\bbchar} % double stroked letters \hspace{1cm} ABC\text{-}XYZ
\texttt{\bf} % sans serif bold \hspace{1cm} abc\text{-}xyz, ABC\text{-}XYZ
\texttt{\bi} % sans serif bold slanted \hspace{1cm} abc\text{-}xyz, ABC\text{-}XYZ

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

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

1.4 Typical elements of the document

1.4.1 Chapters and sections

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

\tit Document title \langle end of line \rangle
\chap Chapter title \langle end of line \rangle
\sec Section title \langle end of line \rangle
\secc Subsection title \langle end of line \rangle

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

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

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

If a title is so long then it breaks into more lines in the output. It is better to hint at the breakpoints because \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\nlJ 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 &\equiv \sqrt{a^2+b^2} \& \eqmark \cr}

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

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

<table>
<thead>
<tr>
<th>age</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–1</td>
<td>unmeasured</td>
</tr>
<tr>
<td>1–6</td>
<td>observable</td>
</tr>
<tr>
<td>6–12</td>
<td>significant</td>
</tr>
<tr>
<td>12–20</td>
<td>extremal</td>
</tr>
<tr>
<td>20–40</td>
<td>normal</td>
</tr>
<tr>
<td>40–60</td>
<td>various</td>
</tr>
<tr>
<td>60–∞</td>
<td>moderate</td>
</tr>
</tbody>
</table>

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

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

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

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

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

There are five prepared counters A, B, C, D and E. They are reset in each chapter and section\(^3\). They can be used in context of `\numberedpar ⟨letter⟩{⟨text⟩}` macro. For example:

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

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

---

\(^3\) This feature can be changed, see the section 2.26 in the technical documentation.
1.4.3 References

Each automatically numbered object documented in sections 1.4.1 and 1.4.2 can be referenced if optional parameter \[⟨label⟩\] 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⟩] or \pgref[⟨label⟩]. 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].

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⟩]\wlabel{⟨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 \_tocborder, \_pgborder, \_citeborder, \_refborder and \_urlborder as the triple of RGB components of the used color. Example:

\def\_tocborder {1 0 0} % links in table of contents: red frame
\def\_pgborder {0 1 0} % links to pages: green frame
\def\_citeborder {0 0 1} % links to references: blue frame
By default, these macros are not defined. It means that no frames are created.

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

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

The \url macro prints its parameter in \tt font and creates a potential breakpoints in it (after slash or dot, for example). If the \hyperlinks declaration is used then the parameter of \url is treated as an external URL link. An example: \url{http://www.olsak.net} creates http://www.olsak.net. The characters \%, \, \#, \{, and \} have to be protected by backslash in the \url argument, the other special characters \~, \^, \& can be written as single character4. 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 by escaped in the \langle url\rangle 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), ...
\style A % items of type A), B), C), ...
\style x % small rectangle
\style X % big rectangle

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

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

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

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

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

1.4.6 Tables

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

generates the result:

<table>
<thead>
<tr>
<th>Month</th>
<th>commodity</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>notebook</td>
<td>$ 700</td>
</tr>
<tr>
<td>February</td>
<td>skateboard</td>
<td>$ 100</td>
</tr>
<tr>
<td>July</td>
<td>yacht</td>
<td>k$ 170</td>
</tr>
</tbody>
</table>

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

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

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

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

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

The \texttt{\tskip\{dimen\}} command works like the \texttt{\noalign{\vskip\{dimen\}}} immediately after \texttt{\cr*} commands but it doesn’t interrupt the vertical lines. You can use the following parameters for the \texttt{\table} macro. Default values are listed too.

\begin{verbatim}
\everytable={} % code used in \vbox before table processing
\thistable={} % code used in \vbox, it is removed after using it
\tabiteml={\enspace} % left material in each column
\tabitemr={\enspace} % right material in each column
\tabstrut={\strut} % strut which declares lines distance in the table
\tablinespace=2pt % additional vert. space before/after horizontal lines
\vvkern=1pt % space between lines in double vertical line
\hhkern=1pt % space between lines in double horizontal line
\tabskip=0pt % space between columns
\tabskipl=0pt \tabskipr=0pt % space before first and after last column
\end{verbatim}

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

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

\begin{verbatim}
\frame{\table{\{c|l||l|r\}\{\cr \mspan3[\{c\}]\bf Title\} \cr l \noalign{\kern\hhkern}\cr li first & second & third \cr li seven & eight & nine \cr li}}
\end{verbatim}
creates the following result:

<table>
<thead>
<tr>
<th>Title</th>
<th>first</th>
<th>second</th>
<th>third</th>
</tr>
</thead>
<tbody>
<tr>
<td>seven</td>
<td>eight</td>
<td>nine</td>
<td></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}
\thispagestyle{myheadings}
\table{|8{c|}}{
\vrule height 20pt depth10pt width0pt
\baselineskip=20pt \tablinespace=0pt \rulewidth=.8pt
\\hline
\mspan2[c]{\{\text{Singular}\}} & \mspan3[c]{\{\text{Plural}\}} \\
\mspan2[c]{\{} & \mspan3[c]{\{} \& \mspan3[c]{\{} \& \mspan3[c]{\{} \\
\mspan2{I} & \{\text{Inclusive}\} & \mspan3[c]{\{} \& \mspan3[c]{\{} \& \mspan3[c]{\{} \& \mspan3[c]{\{} \\
\mspan2{II} & \{\text{Informal}\} & \mspan3[c]{\{} \& \mspan3[c]{\{} \& \mspan3[c]{\{} \& \mspan3[c]{\{} \\
\mspan2{III} & \{\text{Informal}\} & \mspan3[c]{\{} \& \mspan3[c]{\{} \& \mspan3[c]{\{} \& \mspan3[c]{\{} \\
\hline
\end{verbatim}

You can use \texttt{\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}s and \texttt{\hrule}s can be set by the command \texttt{\rulewidth=\{dimen\}}. The default value given by TEX is 0.4 pt.

The \texttt{\vrule}s in the example above are in the center of the lines. More about it is in technical documentation in section 2.30.5. See the definition of the \texttt{\_tabledeclare} macro, for example.

The \texttt{:} 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 \url{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 text 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. OpTEX sets \texttt{\ttline=-1} by default.

The indentation of each line in display verbatim is controlled by \texttt{\ttindent} register. This register is set to the \texttt{\parindent} by default. Users can change the values of the \texttt{\parindent} and \texttt{\ttindent} independently.

The \texttt{\begtt} command starts the internal group in which the catcodes are changed. Then the \texttt{\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{@[..]*)&$} ... prints @[..]*)&$ without escaping, but you can escape these characters too, if you want.
\code{a \ b} ... two spaces between a b, the second must be escaped
\code{xy\{z} ... xy{z ... unbalanced brace must be escaped
\code{\^M} ... prints \^M, the second ^ must be escaped

You can print verbatim listing from external files by the \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 \hspace=20cm (~) program.c.

The \ttline influences the line numbering by the same way as in \begtt...\endtt environment. If \ttline=-1 then real line numbers are printed (this is the default). If \ttline<=-1 then no line numbers are printed.
The \verbinput can be controlled by \everytt, \ttindent just like in \begtt...\endtt.

The \begtt...\endtt pair or \verbinput can be used for listings of codes. Automatic syntax highlighting is possible, for example \begtt \hisyntax{C} activates colors for C programs. Or \verbinput \hisyntax{HTML} (-) file.html can be used for HTML or XML codes. OpTEX implements C, 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 131) 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
\regmacro \{\def\mylogo\{hbox{\it My}/\LoGo}\} % used in running heads
\regmacro \{\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:

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

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:

\begin{verbatim}
\ii linear/space,vector/space,@
\end{verbatim}

is equivalent to:

\begin{verbatim}
\ii linear/space,vector/space \ii space/linear,space/vector
\end{verbatim}

If you really need to insert the space into the index entry, write `-`

The `\ii` or `\iid` commands can be preceded by `\iiitype \langle letter \rangle`, then such reference (or more references generated by one `\ii`) has the specified type. The page numbers of such references should be formatted specially in the index. \TeX\ implements only `\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 \TeX\book index is a good example.

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

\begin{verbatim}
\begmulti \langle number of columns \rangle \\
\langle text \rangle \\
\endmulti
\end{verbatim}

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

\begin{verbatim}
\sec Index \\
\begmulti 3 \makeindex \endmulti
\end{verbatim}

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

\begin{verbatim}
The \ii chiquadrat $\chi$-quadrat method is ... \\
If the \ii relax `relax` command is used then \TeX/ is relaxing. \\
...
\iiis chiquadrat \{$\chi$-quadrat\}
\iiis relax \{\code{\relax}\}
\end{verbatim}

The `\iiis \langle equivalent \rangle \{\langle text \rangle\}` creates one entry in the “dictionary of the exceptions”. The sorting is done by the `\langle equivalent \rangle` but the `\langle text \rangle` is printed in the index entry list.

The sorting rules when `\makeindex` runs depends on the current language. See section 1.7.1 about languages selection.
1.5.3 Bib\TeX{}ing

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

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

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

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

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

\def\cite\[#1\]{(\rcite\[#1\])} % \cite\[⟨label⟩\] creates (27)
\def\cite\[#1\]{$^\{\rcite\[#1\]\}$} % \cite\[⟨label⟩\] creates^{27}

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 \bib\[⟨label⟩\] commands.
- By \usebib\/{⟨type⟩} \{⟨style⟩\} \{⟨bib-base⟩\} command which reads *.bib files directly.

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

References created manually using \bib\[⟨label⟩\] command.

\bib [tst] P. Olšák. \emph{Typografický systém \TeX{}}.
\hspace{2cm} 269-s. Praha: CSTUG, 1995.

If you are using \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:

\bib [tbn] = \{Olšák, 2001\}

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

\usebib/c \{⟨style⟩\} \{⟨bib-base⟩\} % sorted by \cite-order (c=cite),
\usebib/s \{⟨style⟩\} \{⟨bib-base⟩\} % sorted by style (s=style).
% 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 bib-⟨⟨style⟩⟩.opm where the formatting of the references list is defined. \Op\TeX{} 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, ...
\bibnum=0 \usebib/c (simple) file.bib % generates BB bib-list numbered 1, 2, ...
\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, ...
```

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

OpTeX provides a small number of color selectors: `\Blue`, `\Red`, `\Brown`, `\Green`, `\Yellow`, `\Cyan`, `\Magenta`, `\White`, `\Grey`, `\LightGrey`, and `\Black`. User can define more such selectors by setting four CMYK components or three RGB components. For example

```
\def \Orange {\setcmykcolor{0 0.5 1 0}}
\def \Purple {\setrgbcolor{1 0 1}}
```

The command `\morecolors` reads more definitions of color selectors from the BTeX file `x11named`. There are about 300 color names like `\DeepPink`, `\Chocolate` etc. If there are numbered variants of the same name, then the letters B, C, etc. are appended to the name in OpTeX. For example `\Chocolate` is Chocolate1, `\ChocolateB` is Chocolate2 etc.

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

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

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

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

The linear combination is done in CMYK subtractive color space by default (RGB colors used in `\colordef` argument are converted first). If the resulting component is greater than 1 then

---

5 Printed output is more equal to the monitor preview especially if you are using ICC profile for your printer.
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 \texttt{\rgbcolordef} instead of \texttt{\colordef} if you want to mix colors in the additive RGB color space.

The following example defines the macro for the colored text on the colored background.

Usage: \texttt{\coloron\langle background\rangle\langle foreground\rangle\{\langle text\rangle\}}

The \texttt{\coloron} can be defined as follows:

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

\texttt{\coloron\Yellow\Brown\{The brown text on the yellow background\}}

1.6.2 Images

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

The \texttt{\picwidth} is an equivalent register to \texttt{\picw}. Moreover, there is an \texttt{\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 \texttt{\picdir}. This token list is empty by default, this means that the image files are searched in the current directory. Example: \texttt{\picdir=\{img/\}} supposes that image files are in \texttt{img} subdirectory. Note: the directory name must end by / in the \texttt{\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. Op\TeX{} supports this feature by \texttt{\inkinspic\{\langle filename\rangle.pdf\}} command. It reads and displays both: PDF image and labels generated by Inkscape.

If you want to create vector graphics (diagrams, schema, geometry skicing) then you can do it by Wysiwyg graphics editor (Inkscape, Geogebra for example), export the result to PDF and include it by \texttt{\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 \texttt{\pdfsetmatrix\{\langle a\rangle\ \langle b\rangle\ \langle c\rangle\ \langle d\rangle\}} command makes the internal multiplication with the current matrix so linear transformations can be composed. One linear transformation given by the \texttt{\pdfsetmatrix} above transforms the vector \([0, 1]\) to \([\langle a\rangle, \langle b\rangle]\) and \([1, 0]\) to \([\langle c\rangle, \langle d\rangle]\). The stack-oriented commands \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\rlap{\langle transformed text\rangle}}\texttt{\pdfrestore} or something similar is recommended.

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

\begin{verbatim}
\pdfscale{\langle horizontal-factor\rangle}{\langle vertical-factor\rangle}
\pdfrotate{\langle angle-in-degrees\rangle}
\end{verbatim}

These macros simply call the properly \texttt{\pdfsetmatrix} command.

\textsuperscript{6} A powerful and free Wysiwyg editor for creating vector graphics.
\textsuperscript{7} Chose “Omit text in PDF and create LaTeX file” option.
It is known that the composition of transformations is not commutative. It means that the order is important. You have to read the transformation matrices from right to left. Example:

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

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

You can see that \TeX{} knows nothing about dimensions of transformed material, it treats it as with a zero dimension object. The \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, \pdfscale, \pdfrotate} with their parameters. The \texttt{⟨text⟩} is transformed text.

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

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
\ignoremargins=N % ignore margins by surrounding text
\hhkern=0pt \vvkern=0pt} % left-right margin, top-bottom margin
\end{verbatim}

The total distance from text to oval boundary is \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⟩ (color, size, font etc.), put such setting right in front of the ⟨text⟩: \texttt{\inoval[⟨text settings⟩]{⟨text⟩}}.

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

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

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

\begin{verbatim}
\clipincircle 3cm 3.5cm 6cm 7cm \picw=6cm \inspic{myphoto.jpg}
\end{verbatim}
1.6.5 Putting images and texts wherever

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

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

1.7 Others

1.7.1 Using more languages

\TeX\ is prepared to use hyphenation patterns for all languages if such patterns are available in your \TeX\ system. Only U.S. English patterns (original from Plain \TeX) are preloaded. Hyphenation patterns of all other languages are loaded on demand when you first use the \langle iso-code\rangle \lang\ command in your document. For example \delang for German, \cslang for Czech, \pllang for Polish. The \langle iso-code\rangle is a shortcut of the language name (mostly from ISO 639-1). You can list all available languages by \langlist\ macro. This macro prints now:

en(U.S.english) enus(U.S.englishmax) engb(U.K.english) it(italian) id(interlingua) id(indonesian) es(czech) sk(slovak)
de(n.German) fr(French) pl(Polish) cy(Welsh) da(Danish) es(Spanish) sl(Slovenian) fi(Finnish) hu(Hungarian)
tr(Turkish) et(Estonian) eu(Esperanto) ga(Irish) nb(Bokmal) nn(Nynorsk) nl(Dutch) pt(Portuguese) ro(Romanian)
hr(Croatian) zh(Pinyin) is(Icelandic) hus(Upsorbian) al(Afrikaans) gl(Galician) kmr(Kurmanji) tk(Turkmen)
lc(Latvian) lac(classic Latin) lsl(liturgical Latin) elm(mono Greek) elp(Greek) grec(ancient Greek) ca(Catalan)
cop(Coptic) mn(Mongolian) sa(Sanskrit) ru(Russian) uk(Ukrainian) ar(Arabic) hi(Hindi) kn(Kannada) lv(Latvian)
la(Latin) ml(Malayalam) mr(Marathi) or(Oriya) pa(Punjabi) ta(Tamil) te(Telugu) be(Belarusian) bg(Bulgarian)
bn(Bengali) cu(Churchslavonic) deo(old German) gsw(swiss German) eo(Esperanto) fr(French) gu(Gujarati)
ka(Georgian) mk(Macedonian) oc(Occitan) pi(Pali) pms(Piedmontese) rm(Romansh) sr(Serbian) sv(Swedish)
th(Thai) et(Thai) ethi(Ethiopic)
fis(school Finnish)

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

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

You can switch between language patterns by \langle iso-code\rangle \lang\ commands mentioned above. Default is \enlang.

\TeX\ generates three phrases used for captions and titles in technical articles or books: “Chapter”, “Table” and “Figure”. These phrases need to be known in used language and it depends on the previously used language selectors \langle iso-code\rangle \lang. \TeX\ declares these words only for few languages: Czech, German, Spanish, French, Greek, Italian, Polish, Russian, Slovak and English, If you need to use these words in other languages or you want to auto-generate more words in your macros, then you can declare it by \sdef or \_langw commands as shown in section 2.37.3.

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

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

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

\address
  ⟨first line of address⟩
  ⟨second line of address⟩
  ⟨etc.⟩
  ⟨empty line⟩

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

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

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

1.7.3 Loading other macro packages

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

\load [qrcode, tikz]

does \input qrcode.opm and \input tikz.tex and it saves local information about the fact that these file names qrcode and tikz were already used, i.e. next \load will skip them.

It is strongly recommended to use the \load macro for loading external macros if you need them. On the other hand, if your source document is structured to more files (with individual chapters or sections), use simply the \input primitive.
The macro packages intended to OpTeX have the name *.opm. The following packages are distributed as part of OpTeX:

- `qrcode.opm` enables to create QR codes.
- `vlna.opm` enables to protect of one-letter prepositions and more things automatically.
- `emoji.opm` defines \emoji{⟨name⟩} command for colored emoticons.
- `plain-at.opm` defines the old names from plain TeX.

See the directory `optex/pkg/` and these files for more information about them.

### 1.7.4Lorem ipsum dolor sit

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

### 1.7.5Logos

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

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

### 1.7.6 The last page

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

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

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

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

### 1.7.7 Use OpTEX

The command \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
\ii item1,item2 % insertion the items to the index
\makeindex % the index is generated
\label [labname] % link target location
```
1.9 API for macro writers

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

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

If you are writing a document or macros specific for the document, then use simply user namespace (\texttt{\foo}). If you are writing more general macros, then use private namespace (\texttt{\_foo}), but you should declare your own namespace by \texttt{\namespace} macro and you have to follow the naming discipline described in section 2.2.4.
The alphabetically sorted list of macros typically usable for macro writers follows. More information about such macros can be found in the technical documentation. You can use hyperlinks here in order to go to the appropriate place of the technical documentation.

\addto \macro{\text} adds \text at the end of \macro body.
\def \char{\body} defines \char active character with meaning \body.
\afterfi \text expands \fi to \text.
\bp \(\dimen\expression\) expands \TeX dimension to decimal number in \bp without unit.
\codedecl \sequence \{\info\} is used at beginning of macro files.
\colordef \macro \{\mix\ of \colors\} declares \macro as color switch.
\cs \{\string\} expands \text{string}.
\doc ... \cod encloses documentation text in the macro code.
\olddef \macro \#\{\body\} defines \macro with parameter separated to end of line.
\endcodeline closes the part of macro code in macro files.
\endnamespace closes name space declared by \namespace.
\eqbox \{\label\} \{\text\} creates \box \{\text\} with common width across whole document.
\expr \{\expression\} expands to result of the \expression with decimal numbers.
\fontdef \f \{\font spec\} declares \f as font switch.
\fontlet \fa=\fb \{\sizespec\} declares \fa as the same font switch like \fb at given \sizespec.
\foreach \list \do \{\parameters\} \{\what\} is exapandable loop over \list.
\foreachdef \macro \{\parameters\} \{\what\} declares expandable \macro as loop over \list.
\fornum \from \to \do \{\what\} is expandable loop with numeric variable.
\incr \{\counter\} increases and \decr \{\counter\} decreases \{\counter\} by one globally.
\ignoreit \{\one}, \ignoresecond \{\one\} \{\two\} ignores given parameter.
\expandafter \ignorept the \{\dimen\} expands to decimal number \{\dimen\} without pt.
\isempty, \listisempty, \isequal, \ismacro, \isdefined, \islist, \isfile, \isfont do various tests. Example: \islist \list \{\text\} \istrue does \iftrue if \text is in \list.
\isnextchar \char \{\text1\} \{\text2\} performs \text1 if next character is \char, else \text2.
\kv \{\key\} expands to value when key-value parameters are used.
\loop ... \repeat is classical Plain \TeX loop.
\mathstyles \{\math list\} enables to create macros dependent on current math style.
\namespace \{\pk\} declares name space used by package writers.
\newcount, \newdimen etc. are classical Plain \TeX allocators.
\newif \iffoo declares boolean \iffoo as in Plain \TeX.
\newif \iffoo declares boolean \iffoo.
\input \{\filename\} reads file like \input but with standard catcodes.
\optdef \macro \{\opt-default\} \{\parameters\} \{\body\} defines \macro with \opt parameter.
\opwarning \{\text\} prints \text to the terminal and .log file as warning.
\private \sequence \sequence ... ; declares \sequence for private name space.
\public \sequence \sequence ... ; declares \sequence for public name space.
\readkv \macro reads parameters from \macro in key-value format.
\replstring \macro\{\stringA\} \{\stringB\} replaces all \text{stringA} to \text{stringB} in \macro.
\sdef \string \{\parameters\} \{\body\} behaves like \sdef \string \{\parameters\} \{\body\}.
\settable and \restoretable manipulate with stack of catcode tables.
\set \{\string\} \{\stringB\} behaves like \set \{\string\}=\{\stringB\}.
\xdef \string \{\parameters\} \{\body\} behaves like \xdef \string \{\parameters\} \{\body\}.
\trycs \{\string\} \{\text\} expands \text if it is defined else expands \{text\}.
\useit \{\one\}, \usessecond \{\one\} \{\two\} uses given parameter.
\wlog \{\text\} writes \text to .log file.
\wterm \{\text\} writes \text to the terminal and .log file.
\xargs \{\what\} \{\token\} \{\token\} ... ; repeats \what \token for each \token.
1.10 Compatibility with Plain TeX

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

All primitives and common macros have two control sequences with the same meaning: in prefixed and unprefixed form. For example \hbox is equal to _\hbox. Internal macros of OpTeX 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). User should re-define unprefixed forms of control sequences without worries that something internal will be broken (only the sequence \par cannot be re-defined without change of internal TeX behavior because it is hard-coded in TeX, unfortunately).

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

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

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

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

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

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

The \sec macro is reserved for sections but original Plain TeX declares this control sequence for math secans.

\footnote{The math accents macros like \acute, \bar, \dot, \hat still work.}
Chapter 2
Technical documentation

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

luatex -ini optex.ini

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

optex optex-doc.tex

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

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

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

2.1 The main initialization file

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

Category codes are set first. Note that the _ is set to category code “letter", it can be used as a part of control sequence names. Other category codes are set as in plain \TeX.

The \optexversion and \fmtname are defined.

We check if Lua\TeX engine is used at -ini state. And the ^^J character is set as \newlinechar.
The basic macros for macro file syntax is defined, i.e. \_endcode, \_doc and \_cod. The \_codedecl will be re-defined later.

Individual *.opm macro files are read.

The file optex.lua is embedded into the format as byte-code. It is documented in section 2.39.
The \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 \OPEX are used and defined with _ prefix. The user can be sure that when he/she does \def\foo then neither internal macros of Op\TeX\ nor \TeX\ primitives will be damaged. For example \def\if{...} will not damage macros because Op\TeX\’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.

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

We never change catcode of _, so internal macros can be redefined by user without problems if it is desired. We don’t need something like \makeatletter from \LaTeX. Op\TeX\ 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 Op\TeX, 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 unprefixed variant of a control sequence is declared already but we need the prefixed variant too.

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

2.2.2 Namespace of control sequences for users

Users can define or declare any control sequence with a name without any _. This does not make any problem. Only one exception is the reserved control sequence \par. It is generated by the tokenizer (at empty lines) and used as internal in \TeX.

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, Op\TeX internal macros and packages internal macros.
- Multiletter control sequences in the form \⟨word⟩_ or \⟨word⟩_⟨one-letter⟩, where ⟨word⟩ is a sequence of letters, are inaccessible, because they are interpreted as \⟨word⟩ followed by _ or as \⟨word⟩ followed by _⟨one-letter⟩. This is important for writing math, for example:

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

- Control sequences in the form \langle pkg\rangle\_\langle word\rangle is intended for package writers as internal macros for a package with \langle pkg\rangle 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 OpTeX macros is stored in one file with .opm extension (means OPtex Macros). Your local macros should be in a normal *.tex file.

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

\_codedecl \sequence {Name <version>}

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

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

### 2.2.4 Name spaces for package writers

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

The package writer does not need to write repeatedly \_\langle pkg\rangle\_foo \_\langle pkg\rangle\_bar etc. again and again in the macro file.¹ When the \_namespace {\langle pkg\rangle} is declared at the beginning of the macro file then all occurrences of \_\langle pkg\rangle\_foo will be replaced by \_\langle pkg\rangle\_foo at the input processor level. The macro writer can write (and backward can read his/her code) simply with \_\langle pkg\rangle\_foo, \_\langle pkg\rangle\_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 = \_\langle pkg\rangle\_foo when \_namespace {\langle pkg\rangle} 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 namespace by \_nspublic \macro;

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

If the package writer needs to declare a control sequence by \_newifi\_if\langle pkg\rangle\_bar, for example \_newifi\_ifqr_incorner. Then the control sequences \_qr_incornertrue and \_qr_incornerfalse can be used (or the sequences \_incornertrue and \_incornerfalse when \_namespace{qr} is used).

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

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

- Don’t use control sequences from the user namespace in the macro bodies if there is no explicit and documented reason to do this.

¹ We have not adopted the idea from expl3 language:)

33
• Use control sequences from OpTEX and primitive namespace in read-only mode, if there is not an explicit and documented reason to redefine them.
• Use \_⟨pkg⟩⟨name⟩ for your internal macros or \.⟨name⟩ if the \_namespace{⟨pkg⟩} is declared. See section 2.2.4.
• Use \load (or better: \_load) for loading more external macros if you need them. Don’t use \_input explicitly in such cases. The reason is: the external macro file is not loaded twice if another macro or the user needs it explicitly too.
• Use \_codedecl as your first command in the macro file and \_endcode to close the text of macros.
• You can write more documentation after the \_endcode command.
• The OpTEX catcodes are set when \load your package (i.e. plain TEX catcodes plus catcode of _ is 11). If a catcode is changed during loading your package then it is forgot because \load returns to catcodes used before loading package. If you want to offer a catcode changing for users then insert it to a macro which can be used after loading.

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

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

### 2.2.6 The implementation of the namespaces

```latex
\_checkexists #1#2#3{\_unless \_ifcsname #2\csstring#3\endcsname \_errmessage {\_string#1: \_bslash#2\csstring#3 must be declared}\_fi}
```

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

### prefixed.opm

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

```latex
\_checkexists \_where ⟨prefix⟩⟨sequence⟩ prints error if the control sequence propagated to a new name space by \_public etc. macros is not declared.
\_xargs ⟨what⟩ ⟨sequence⟩ ⟨sequence⟩ ... \_def \_private \_public \_xargs \_ea ;
```

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

The \_endcode is defined as \endinput in the optex.ini file. \wterm {⟨text⟩} prints ⟨text⟩ to the terminal and 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.
2.4 Basic macros

We define first bundle of basic macros.

\let_bgroup={ \let_egroup=}
def \empty {} 
def \space {} 
def \null {} 
def \wlog {\immediate\write-1} % write on log file (only)

\public \bgroup \egroup \empty \space \null \wlog ;
\_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.

\_long\_def \bslash \{\csstring\\\} \_long\_def \nbb \{\bslash\bslash\} \_long\_def \pcent \{\csstring\%\}

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

\_long\_def \sdef #1{\_ea\_def \_csname#1\_endcsname} \_long\_def \sxdef #1{\_ea\_xdef \_csname#1\_endcsname} \_long\_def \slet #1#2{\_ea\_let \_csname#1\_ea\endcsname \_csname#2\_endcsname}

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

\_long\_def \adef #1{\_catcode`#1=13 \_begingroup \_lccode`~=`#1\_lowercase{\_endgroup\_def~}}

\_long\_def \addto \macro\{\text\} adds \text to your \macro, which must be defined.

\_long\_def \addto \#1\#2{\_ea\_def \_ea\#1\_ea\{\#1\#2\}}

\incr\{\counter\} increases \counter by one globally. \decr\{\counter\} decreases \counter by one globally.

\_long\_def \incr \#1{\_global\_advance\#1by1} \_long\_def \decr \#1{\_global\_advance\#1by-1}

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

\_long\_def \opwarning \#1{\_wterm{WARNING l._the_inputlineno: #1.}}

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

\_long\_def \loggingall\{\_tracingcommands=3 \_tracingstates=2 \_tracingpages=1 \_tracingoutput=1 \_tracinglostchars=1 \_tracingmacros=2 \_tracingparagraphs=1 \_tracingrestores=1 \_tracingcantokens=1 \_tracingifs=1 \_tracinggroups=1 \_tracingassigns=1\} \_long\_def \tracingall\{\_tracingonline=1 \_loggingall\}

Write a warning if the user did not load a Unicode Font or if there were unresolved references. \byehook is used in the \bye macro.
2.5 Allocators for \TeX registers

Like plain\TeX, the allocators \texttt{\newcount}, \texttt{\newwrite}, etc. are defined. The registers are allocated from 256 to the \texttt{\_mai⟨type⟩} which is 65535 in Lua\TeX.

Unlike in Plain\TeX, the mentioned allocators are not \texttt{\outer}.

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

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

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

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

The limits are set first.

\begin{verbatim}
\chardef\_maicount = 65535 % Max Allocation Index for counts registers in Lua\TeX
\let\_maidimen = \_maicount
\let\_maiskip = \_maicount
\let\_maimuskip = \_maicount
\let\_maibox = \_maicount
\let\_maitoks = \_maicount
\chardef\_mairead = 15
\chardef\_maiwrite = 15
\chardef\_maifam = 255
\end{verbatim}

Each allocation macro needs its own counter.

\begin{verbatim}
\countdef\_countalloc=10 \_countalloc=255
\countdef\_dimenalloc=11 \_dimenalloc=255
\countdef\_skipalloc=12 \_skipalloc=255
\countdef\_muskipalloc=13 \_muskipalloc=255
\countdef\_boxalloc=14 \_boxalloc=255
\countdef\_toksalloc=15 \_toksalloc=255
\countdef\_readalloc=16 \_readalloc=1
\countdef\_writealloc=17 \_writealloc=1
\countdef\_famalloc=18 \_famalloc=3
\end{verbatim}

The common allocation macro \texttt{\_allocator \⟨sequence⟩ \langle⟨type⟩⟩ \langle⟨primitive declarator⟩⟩} is defined. This idea was used in classical plain \TeX by Donald Knuth too but the macro from plain \TeX seems to be more complicated:).
The allocation macros \newcount, \newdimen, \newskip, \newmskip, \newbox, \newtoks, \newread, \newwrite and \newfam are defined here.

The \newinsert macro is defined differently than others.

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

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 \specialcategory dance, so \z@skip, \p@, etc. are not defined in Op\TeX. If you need such control sequences then you can initialize them by \load[plain-at]. Only the \zo and \zoskip (equivalents to \z@ and \z@skip) are declared here and used in some internal macros of Op\TeX for improving speed.
2.6 If-macros, loops, is-macros

2.6.1 Classical \newif

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

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

\newif \afterfi {⟨what to do⟩}(⟨ignored⟩)\fi closes condition by \fi and processes ⟨what to do⟩. Usage:
\if<something> \afterfi{⟨result is true⟩} \else \afterfi{⟨result is false⟩} \fi

2.6.2 Loops

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

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

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

\foreach ⟨list⟩\do {⟨what⟩} repeats ⟨what⟩ for each element of the ⟨list⟩. The ⟨what⟩ can include #1 which is substituted by each element of the ⟨list⟩. The macro is expandable.
\foreach ⟨list⟩\do ⟨parameter-mask⟩{⟨what⟩} reads parameters from ⟨list⟩ repeatedly and does ⟨what⟩ for each such reading. The parameters are declared by ⟨parameter-mask⟩. Examples:
\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 \langle list \rangle \do \langle what \rangle is equivalent to \foreach \langle list \rangle \do #1\{\langle what \rangle \}.

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

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

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

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

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

Implementation note: we cannot use \TeX-groups for nesting levels because we want to do the macros expandable. We must implement a special for-stack which saves the data needed by \foreach and \fornum. The \_putforstack is used when \for* is initialized and \_getforstack is used when the \for* macro ends. The \_forlevel variable keeps the current nesting level. If it is zero, then we need not save nor restore any data.
User can define own expandable “foreach” macro by \foreachdef \macro \langle parameter-mask \rangle \{ \langle what \rangle \} which can be used by \macro \{ \langle list \rangle \}. The macro reads repeatedly parameters from \langle list \rangle using \langle parameter-mask \rangle and does \langle what \rangle for each such reading. For example \foreachdef \mymacro #1,\{\#1\}\mymacro{a,b,cd,efg,} expands to [a][b][cd][efg]. Such user defined macros are more effective during processing than \foreach itself because they need not to operate with the for-stack.

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.

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

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

\isdefined \{\langle csname \rangle\}\iftrue is true if \langle csname \rangle is defined. The macro is expandable.
\islist \list{⟨text⟩}\iftrue is true if the ⟨text⟩ is included the macro body of the \list. The category codes are relevant here. The macro is not expandable.

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

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

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

2.7 Setting parameters

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

- primitive registers used in build-in algorithms of \TeX,
- registers declared and used by \OPTEX macros.

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

The registers are represented by their names (control sequences). If the user re-defines this control sequence then the appropriate register exists steadily and build-in algorithms are using it without change. But user cannot access its value in this case. \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:

\begin{verbatim}
10 \_parindent=20pt % indentation of paragraphs
11 \_pretolerance=100 % parameters used in paragraph breaking algorithm
12 \_tolerance=200
13 \_hbadness=1000
14 \_vbadness=1000
15 \_doublehyphenpenalty=10000
16 \_finalhyphenpenalty=5000
17 \_adjhyphenpenalty=10000
18 \_uchyph=1
19 \_defaulthyphenchar=`\-
20 \_defaultskewchar=-1
21 \_hfuzz=0.1pt
22 \_vfuzz=0.1pt
23 \_overfullrule=5pt
24 \_linepenalty=10 % penalty between lines inside the paragraph
25 \_hyphenpenalty=50 % when a word is broken
26 \_ezhyphenpenalty=50 % when the hyphenmark is used explicitly
27 \_binoppenalty=700 % between binary operators in math
28 \_relpenalty=500 % between relations in math
29 \_brokenpenalty=100 % after lines if they end by a broken word.
30 \_displaywidowpenalty=50 % before last line of paragraph if display math follows
31 \_predisplaypenalty=10000 % above display math
32 \_postdisplaypenalty=0 % below display math
33 \_delimeterfactor=901 % parameter for scaling delimiters
34 \_predisplaypenalty=10000 % above display math
35 \_postdisplaypenalty=0 % below display math
36 \_scriptspace=0.5pt
37 \_maxdepth=4pt
38 \_splitmaxdepth\_maxdimen
39 \_boxmaxdepth\_maxdimen
40 \_parskip=0pt plus 1pt
41 \_abovedisplayskip=12pt plus 3pt minus 9pt
42 \_abovedisplayshortskip=0pt plus 3pt minus 9pt
43 \_belowdisplayskip=12pt plus 4pt minus 4pt
44 \_belowdisplayshortskip=7pt plus 3pt minus 4pt
45 \_parfillskip=0pt plus 1fil
46 \_thinmuskip=3mu
47 \_medmuskip=4mu plus 2mu minus 4mu
48 \_thickmuskip=5mu plus 5mu
\end{verbatim}

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

\begin{verbatim}
56 \_topskip=10pt % top edge of page-box to first baseline distance
57 \_splittopskip=10pt
\end{verbatim}

2.7.2 Plain \TeX registers

Declared registers used in plain \TeX

\begin{verbatim}
65 \_newskip\_smallskipamount \_smallskipamount=3pt plus 1pt minus 1pt
66 \_newskip\_medskipamount \_medskipamount=6pt plus 2pt minus 2pt
67 \_newskip\_bigskipamount \_bigskipamount=12pt plus 4pt minus 4pt
68 \_newskip\_normalbaselineskip \_normalbaselineskip=12pt
69 \_newskip\_normallineskip \_normallineskip=1pt
70 \_newdimen\_normallineskiplimit \_normallineskiplimit=0pt
71 \_newdimen\_jot \_jot=3pt
72 \_newcount\_interdisplaylinepenalty \_interdisplaylinepenalty=100
73 \_newcount\_interfootnotelinepenalty \_interfootnotelinepenalty=100
74 \_def\_normalbaselines\_lineskip=\_normallineskip
75 \_def\_normalbaselineskip\_normalbaselineskip=\_lineskiplimit=\_normallineskiplimit
76 \_def\_frenchspacing\_sfcode`\_=1000 \_sfcode`\?=1000 \_sfcode`\!l=1000
\end{verbatim}
2.7.3 Different settings than in plain TeX

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

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

Different values than in plain TeX have the following primitive registers. We prohibit orphans, set more information for tracing boxes, set page origin to the upper left corner of the paper (no at 1in, 1in coordinates) and set default page dimensions as A4, no letter.

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

2.7.4 OpTeX parameters

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

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

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

\begin{verbatim}
175 \_newtoks\_picdir
176 \_public \picdir ;
\end{verbatim}

You can control the dimensions of included images by the parameters \texttt{\picwidth} (which is equivalent to \texttt{\picw}) and \texttt{\picheight}. By default these parameters are set to zero: the native dimension of the image is used. If only \texttt{\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 \texttt{\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 \texttt{\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 \texttt{\inspic} macros.

\begin{verbatim}
194 \_newdimen\_picwidth \_picwidth=0pt \_let\picw=\_picwidth
195 \_newdimen\_picheight \_picheight=0pt
196 \_public \picwidth \picheight ;
\end{verbatim}

The \texttt{\everytt} is the token list used in \texttt{\begtt...\endtt} environment and in the verbatim group opened by \texttt{\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 \texttt{\begtt} is processed after the \texttt{\everytt}. This code should overwrite \texttt{\everytt} settings. Use \texttt{\everytt} for all verbatim environments in your document and use a code after \texttt{\begtt} locally only for this environment.

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

\begin{verbatim}
219 \_newtoks\_everytt
220 \_newtoks\_everyintt
221 \_public \everytt \everyintt ;
\end{verbatim}

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

\begin{verbatim}
235 \_newcount\_ttline \_ttline=-1 % last line number in \begtt...\endtt
236 \_public \ttline ;
\end{verbatim}

The \texttt{\ttindent} gives default indentation of verbatim lines printed by \texttt{\begtt...\endtt} pair or by \texttt{\verbinput}. The \texttt{\ttshift} gives the amount of shift of all verbatim lines to the right. Despite the \texttt{\ttindent}, it does not shift the line numbers, only the text.
The \texttt{\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 \texttt{\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 \texttt{\ttindent=\parindent} and \texttt{\iindent=\parindent}.
The tabulator "^I" has its category code like space: it behaves as a space in normal text. This is a common plain \TeX setting. But in the multiline verbatim environment it is active and expands to the \hskip{}⟨dimen⟩ where ⟨dimen⟩ is the width of \tabspaces{} spaces. Default \tabspaces{}=3 means that tabulator behaves like three spaces in multiline verbatim.

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

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

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

The \everylist{} tokens list is applied after the group is opened by \begitems{}.

The \ilevel{} keeps the value of the current nesting level of the items list.

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

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

The \begmulti{} and \endmulti{} pair creates more columns. The parameter \colsep{} declares the space between columns. If \n{} columns are specified then we have \n{}−1 \colsep{}s 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 \biboptions{} is used in the iso690 bib-style for global options, see section 2.32.5.

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

\everycapitonf{} is used before printing caption in figures and \everycapitonf{} is used before printing caption in tables.
The \texttt{everyii} tokens list is used before \texttt{noindent} for each Index item when printing the Index.

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

The \texttt{mnotesize} is the horizontal size of the marginal notes.

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

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

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

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

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

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

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

\texttt{tabskip1} is \texttt{tabskip} used before first column, \texttt{tabskipr} is \texttt{tabskip} used after the last column.

\texttt{tsize} is virtual unit of the width of paragraph-like table items when \texttt{table \texttt{pxto(size)}} is used.

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

\texttt{lmfil} is “left matrix filler” (for \texttt{matrix} columns). The default value does centering because the right matrix filler is directly set to \texttt{hfil}.
The output routine uses token list `\headline` and `\footline` in the same sense as in plain \TeX. If they are non-empty then `\hfil` or `\hss` must be here because they are used inside `\hbox to\hsize`.

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

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

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

\[
\begin{align*}
\text{\headlinedist} &= 14\text{pt} \\
\text{\footlinedist} &= 24\text{pt}
\end{align*}
\]

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

\[
\begin{align*}
\text{\pgbottomskip} &= 0\text{pt} \\
\text{\pgbackground} &= \{\\Yellow\ \hrule\ \text{height}\ 0\text{pt}\ \text{depth}\ \pdfpageheight\ \text{width}\ \pdfpagewidth\}
\end{align*}
\]

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

\[
\text{\headline}=\{}\ \text{\nextpages}=\{\text{\headline}=\{\\rmfixed\ \fistmark\ \hfil\}\}
\]

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

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

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

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

Op\TeX defines “Standard Op\TeX markup language”\textsuperscript{2} which lists selected commands from chapter 1 and gives their behavior when a converter from Op\TeX document to HTML or Markdown or \LaTeX is used.

\textsuperscript{2} Will be developed in 2021.
The structure-oriented commands are selected here, but the commands which declare typographical appearance (page layout, dimensions, selected font family) are omitted. More information for such a converter should be given in \cnvinfo{⟨data⟩}. \OpTEX simply ignores this but the converter can read its configuration from here. For example, a user can write:

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

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

2.8 More OpTEX macros

The second bundle of OpTEX macros is here.

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

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

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

The implementation of the catcodetable stack follows.

The current catcodes are managed in the \catcodetable0. If the \setctable is used first (or at the outer level of the stack), then the \catcodetable0 is pushed to the stack and the current table is re-set to the given {⟨catcode table⟩}. The numbers of these tables are stacked to the \_ctablelist macro. The \restorectable reads the last saved catcode table number from the \_ctablelist and uses it.
When a special macro is defined with different catcodes then \normalcatcodes can be used at the end of such definition. The normal catcodes are restored. The macro reads catcodes from \optecatodes table and sets it to the main catcode table 0.

\begin{verbatim}
\_def\_normalcatcodes {\_catcodetable\_optexcatodes \_savecatcodetable0 \_catcodetable0 }
\_public \_normalcatcodes ;
\end{verbatim}

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

\begin{verbatim}
\_def \_load [#1]{\_loadA #1,,
\_end}
\_def \_loadA #1#2,{\_ifx,#1 \_ea \_loadE \_else \_loadB{#1#2}\_ea\_loadA\_fi}
\_def \_loadB #1{\_ifcsname _load:#1\_endcsname \_else
\_isfile {#1.opm}\_iftrue \_opinput {#1.opm}\_else \_opinput {#1}\_fi
\_sxdef{_load:#1}{}%
\_trycs{_afterload}{}\_let\_afterload=\_undefined \_fi
\_def \_loadE #1\_end{\_ea#1\_romannumeral-`\.}
\_newtoks\_opt
\_public \_load ;
\end{verbatim}

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

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

\begin{verbatim}
\_def\_optdef#1[#2]{\_def#1{\_opt={#2}\_isnextchar[{\_cs{_oA:\_string#1}}]{\_cs{_oB:\_string#1\_nospaceafter}}\_sdef{_oA:\_string#1}[##1]{\_opt={##1}\_cs{_oB:\_string#1\_nospaceafter}}\_sdef{_oB:\_string#1\_nospaceafter}\_nospaceafter#1\_def{_optdef#1{\_optdef#1{##1}}}
\_def\_newtoks\_opt
\_public \opt \_optdef ;
\end{verbatim}

The declarator \eoldef\macro \text{[\text{replacement text}]\{\text{params}\}} 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 \text{replacement text}. The catcode of the \endlinechar is reset temporarily when the parameter is scanned.

The macro defined by \eoldef cannot be used with its parameter inside other macros because the catcode dancing is not possible here. But the \bracedparam\macro\{\text{parameters}\} 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.

\begin{verbatim}
\_def\_eoldef #1{\_def #1{\_begingroup \_catcode`\^M=12 \_eoldefA #1}\
\_ea\_def\_csname _\endcsname\_csstring #1:\_endcsname \_catcode`\^M=12 \%
\_catcode`\^M=12 \%
\_def\_eoldefA #1\_endcsname\_csstring #1:\_endcsname \_catcode`\^M=12 \%
\_catcode`\^M=12 \%
\_def\_eoldef #1\_endcsname\_csstring #1:\_endcsname \_catcode`\^M=12 \%
\_catcode`\^M=12 \%
\_public \_eoldef \_skiptoeol \_bracedparam ;
\end{verbatim}

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

The verbatim scanner of \scantoeol keeps category 7 for ` in order to be able to use `~J as comment character which means that the next line continues.

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

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

The \texttt{\catcode} primitive is redefined here. Why? There is very common cases like \texttt{\catcode`~\{something\}} or \texttt{\catcode\^\{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 \{text with spaces\}}\{} expands to \texttt{\{textwithoutspaces\}}.

The \texttt{\_ea\_ignorept\texttt{\{\textdimen\}}} expands to a decimal number \texttt{\textdimen} but without pt unit.

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

\texttt{\_bp{\_dimen}\{3\\_mm\}} \texttt{\_bp\{2\_mm\} m 0 \_bp\{-4\_mm\} 1 S Q}

You can use expandable \texttt{\_expr\{\_expression\}} for analogical purposes. It expands to the value of the \texttt{\_expression} at expand processor level with \texttt{\_decdigits} digits after the decimal point. The \texttt{\_expression} can include \texttt{+/0} and decimal numbers in common syntax.

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

You can define \myframe as follows:

\def\myframedefaults{% defaults:
  frame-color=\Black, % color of frame rules
  text-color=\Black, % color of text inside the frame
  rule-width=0.4pt, % width of rules used in the frame
  margins=2pt, % space between text inside and rules.
}
\optdef\myframe [] #1{%bgROUP
  \ea\addto\ea\myframedefaults\ea{\ea,\the\opt}%
  \makero\myframedefaults
  \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 append the \opt (actual optional parameters) to \myframedefault by \addto macro. Second: we read the parameters by \makero{parameters list} macro. Third: the values can be used by expandable \kv{key} macro. The \kv{key} returns ?? if such key is not declared.

You can use keys without values in the parameters list too, but with additional care. For example, suppose draft option without parameter. If a user writes \myframe [...] 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{%bgROUP
  \ea\addto\ea\myframedefaults\ea{\ea,\the\opt}%
  \replstring\myframedefaults{draft}{DRAFTv=1}%

54
Implementation. The \readkv expands its parameter and does replace-strings in order to remove spaces around equal signs and after commas. Double commas are removed. Then \kvscan reads the parameters list finished by the double comma and saves values to \kv:\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.

\readkv\myframedefaults
...\ifnum\kv(DRAFTv)=1 draft mode\else normal mode\fi...

2.10 Plain \TeX\ macros

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

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

\addto \dospecials{\do\ }

The shortcuts \chardef\@one is not defined in Op\TeX. Use normal numbers instead of such obscurities.

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

Plain \TeX classical \obeylines and \obeyspaces.
Spaces. \thinspace, \negthinspace, \enspace, \enskip, \quad, \qquad, \smallskip, \medskip, \bigskip, \nointerlineskip, \offinterlineskip, \topglue, \vglue, \hglue, \slash.

\protected\def\thinspace{\kern.16667em}
\protected\def\negthinspace{\kern-.16667em}
\protected\def\enspace{\kern.5em}
\protected\def\enskip{\hskip.5em\relax}
\protected\def\quad{\hskip1em\relax}
\protected\def\qquad{\hskip2em\relax}
\protected\def\smallskip{\vskip\smallskipamount}
\protected\def\medskip{\vskip\medskipamount}
\protected\def\bigskip{\vskip\bigskipamount}
\protected\def\nointerlineskip{\prevdepth=-1000pt}
\protected\def\offinterlineskip{\baselineskip=-1000pt\lineskip=0pt\lineskiplimit=\maxdimen}
\protected\def\topglue{\nointerlineskip\vglue-\topskip\vglue}
\protected\def\vglue{\afterassignment\vglA\skip0=}
\protected\def\vglA{\par\dimen0=\prevdepth\hrule\vskip\skip0\vrule\spacefactor=1\vskip\prevdepth=\dimen0}
\protected\def\hglue{\afterassignment\hglA\skip0=}
\protected\def\hglA{\leavevmode\count255=\spacefactor\vrule\width0pt\nobreak\hskip\skip0\spacefactor=\count255}
\protected\def\~{\penalty10000\ }% tie
\protected\def\slash{/\penalty\exhyphenpenalty}% a `/` that acts like a `-`

\public\topglue\vglue\hglue\slash;

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

\protected\def\break{\penalty10000}
\protected\def\nobreak{\penalty10000}
\protected\def\allowbreak{\penalty0}
\protected\def\filbreak{\par\vfill\penalty-200\vfilneg}
\protected\def\goodbreak{\par\penalty-500}
\protected\def\eject{\par\break}
\protected\def\supereject{\par\penalty-20000}
\protected\def\dosupereject{\ifnum\insertpenalties>0% something is being held over
\kern\topskip\penalty-\arraystretch\vfill\supereject\fi}
\protected\def\removelastskip{\ifdim\lastskip=-\z\else\vskip-\lastskip\fi}
\protected\def\smallbreak{\par\ifdim\lastskip<\smallskipamount\removelastskip\penalty-50\smallskip\fi}
\protected\def\medbreak{\par\ifdim\lastskip<\medskipamount\removelastskip\penalty-100\medskip\fi}
\protected\def\bigbreak{\par\ifdim\lastskip<\bigskipamount\removelastskip\penalty-200\bigskip\fi}
\public\break\nobreak\allowbreak\filbreak\goodbreak\eject\supereject\dosupereject\removelastskip\smallbreak\medbreak\bigbreak;

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

\def\line{\hbox to\hsize}
\def\leftline{\hbox{\line(#1\hs)}\hss}
\def\rightline{\hbox{\line(#1\hs)#1}}
\def\centerline{\hbox{\line(#1\hs)#1\hs}\hss}
\def\rlap{\hbox to\zo{#1\hs}}
\def\llap{\hbox to\zo{#1\hs}}
\def\underbar{\hbox{\setbox0=\hbox{#1}\dp0=\z\math\underline{\box0}\$}}
\public\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, \ttraggedright, \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.

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).

\begin{verbatim}
\def \magnification {\afterassignment \magA \count255}
\def \magA {\mag=\count255 \truedimen\hsize \truedimen\vsize \dimen\footins=8truein}
% only for backward compatibility, but \margins macro is preferred.
\public \magnification;
\def \showhyphens #1{\setbox0=\vbox{\parfillskip=0pt \hsize=\maxdimen \tenrm
\pretolerance=-1 \tolerance=-1 \hbadness=0 \showboxdepth=0 \#1}}
\def \bye {\par \vfill \supereject \byehook \end}
\public \showhyphens \bye;
\end{verbatim}

2.11 Preloaded fonts for text mode

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

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

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

\begin{verbatim}
\-codedecl \tenrm {Latin Modern fonts (EC) preloaded <2020-01-23>} % loaded in format
% Only few text fonts are preloaded:
\font\tenrm=ec-lmr10 % roman text
\font\tenbf=ec-lmbx10 % boldface extended
\font\tenit=ec-lmri10 % text italic
\font\tenbi=ec-lmbxi10 % bold italic
\font\tentt=ec-lmtt10 % typewriter
\public \tenrm \tenbf \tenit \tenbi \tentt;
\end{verbatim}
2.12 Scaling fonts in text mode (low-level macros)

2.12.1 The \setfontsize macro

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

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

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

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

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

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

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

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

2.12.2 The \font primitive

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

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

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

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

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

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

2.12.3 The \fontdef declarator

You can declare \texttt{\langle newfont\rangle} by the \texttt{\fontdef} command.

\texttt{\fontdef \langle newfont\rangle \{\langle font modifiers\rangle \langle variant-selector\rangle\}}

\texttt{\example:}
\texttt{\fontdef \bigfont \{\setfontsize{at15pt}\bf\}}

This command runs \texttt{\langle font modifiers\rangle \langle variant-selector\rangle} in an internal group and sets the resulting selected font as \texttt{\langle newfont\rangle}. 
The resulting \texttt{\newfont} declared by \texttt{\fontdef} is “fixed font switch” independent of \texttt{\setfontsize} and other font modifiers. 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 parameter of the \texttt{\fontdef} macro must be exactly finished by the variant selector. More information about font modifiers and variant selectors are in the section 2.13.

### 2.12.4 The \fontlet declarator

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

\begin{verbatim}
\fontlet \langle newfont \rangle = \langle fontswitch \rangle \langle sizespec \rangle
\end{verbatim}

example:

\begin{verbatim}
\fontlet \bigfont = \_tenbf at15pt
\end{verbatim}

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

**Summary:** you can declare font switches:

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

### 2.12.5 Optical sizes

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

There is \texttt{\defaultoptsize=10pt} by default.

Font collections with optical sizes must be registered by the \texttt{\_regtfm} for tfm files or \texttt{\_regoptsizes} for Unicode fonts. OpTeX registers 8bit Latin Modern fonts in the format (\texttt{fonts-resize.opm} file) and OTF Latin Modern fonts in the \texttt{f-lmfonts.opm} file.

### 2.12.6 Implementation notes

The \texttt{\setfontsize \{\textit{sizespec}\}} saves the \texttt{\textit{sizespec}} to the \texttt{\_sizespec} macro. The \texttt{\optsize} value is calculated from the \texttt{\_sizespec}. If the \texttt{\_sizespec} is in the \texttt{\mag{number}} format then the contents of the \texttt{\_sizespec} macro is re-calculated to the \texttt{\at{dimen}} format using previous \texttt{\optsize} value.

\begin{verbatim}
\_newdimen _optsize
_\optsize=10pt
\_newdimen _defaultoptsize
_\defaultoptsize=10pt
\_newdimen _lastmagsize
\_def_\setfontsize #1{%
  _edef_\_sizespec{#1}%
  _ea _\setoptsize _\_sizespec \_relax
  _\reloading
} % reloading
\_def_\setoptsizeA at#1\_relax{_\optsize=#1\_relax\_lastmagsize=_\optsize} % at<dimen>
\_def_\setoptsizeB scaled#1\_relax{_\optsize=_\defaultoptsize\_relax} % scaled<scalenum>
\_def_\setoptsizeC mag#1\_relax{%
  _ifdim _lastmagsize>_zo _\optsize=_lastmagsize
  _else _\optsize=_pdffontsize\font at1
  _\optsize=#1\_optsize
  \_lastmagsize=_\optsize
  _edef_\_sizespec atr\_the\_optsize%
} %
\_public \_\setfontsize \_\defaultoptsize ;
\end{verbatim}
\_resizefont \{\langle variant-name\rangle\}\{\langle font switch\rangle\}, resizes the font given by the variant. The \langle variant-name\rangle is rm or bf or it or bi. The new \langle font-switch\rangle is declared (roughly speaking) by:

\_font \{\langle font switch\rangle\} = \langle font-name of \_ten\langle variant-name\rangle \_sizepec\rangle

If the \_fontselector is defined then it has priority before \langle font-switch\rangle and the font is loaded by:

\_ea\_font = \fontname \_ten\langle variant-name\rangle \_sizepec

The font is loaded by \_doresizefont\{\langle font switch\rangle\}. This macro has meaning \_doresizetfmfont in TFM mode and it switches to \_doresizeunifont when \initunifonts is used.

The \langle font-name of \_initunifonts\rangle is generated differently in OTF mode, see \_doresizeunifont macro.

The \_whatresize is defined as \langle variant-name\rangle.

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

\_def \_fontdef #1#2\%{
  \_ifx \_fontselector \_undefined \def \_fontselector {#1}\fi
  \reloading #2\%
  \_ea \_keepmeaning \_fontselector \_endgroup
}\_def \_fontlet#1#2\%{
  \_ifx #2=\_ea \fontlet \else \_ea \fontlet \_ea\_else \fi
  \_ea \_font \_ea\_fontlet \_ea\_relax \_space \_sizespec \_relax
}\_def \_keepmeaning #1#2\%{
  \_global \_let \_keepmeaningdata=#1\%
}
#2\_let#1=!\_keepmeaningdata \_global\_let\_keepmeaningdata=\_undefined
\_public \fontdef \fontlet);

\newcurrfontsize \langle size spec \rangle sets current font size to the \langle size spec \rangle. It is implemented by \fontlet. The font switch of the current font is extracted by \the\font. We must re-create the control sequence \the\font because its original meaning is set to “inaccessible” by \TeX when \font primitive is started. \resizethefont is implemented by \newcurrfontsize using data from the \sizespec macro.

The variant selector is defined by \protected\def\XX{\_tryloadXX \_tenXX} The \_tryloadXX can be in \relax state if no font modifiers were declared. But normally it does \_resizefont{XX}\tenXX. This meaning is activated by the \reloading macro.

The font selection system allows to use \currvar instead explicitly specified variant selector. The current variant is extracted from \the\font output which could be \_tenXX control sequence. Then \currvar expands to \_rm or \_it etc.

The \_regtfm \langle font id \rangle \langle optical size data \rangle saves the \langle optical size data \rangle concerned to \langle font id \rangle. The \langle optical size data \rangle is in the form as shown below in the code where \_regtfm is used.

The \_whichtfm \langle fontname \rangle expands to the \langle fontname \rangle or to the corrected \langle fontname \rangle read from the \langle optical size data \rangle. It is used in the \rfontskipat macro and it is used in \fontlet macro. It means that each \langle fontname \rangle generated by the \langle fontname \rangle primitive in the \fontlet macro is processed by the \_whichtfm. The real \langle fontname \rangle or corrected \langle fontname \rangle (depending on the optical data does not exist or exist) is the output of the expansion before \font primitive takes this output as its parameter.

The implementation detail: The \_\langle font id \rangle\_reg is defined as the \langle optical size data \rangle and all control sequences \_\langle fontname \rangle:\_reg from this data line have the same meaning because of the \reversetfm macro. The \_whichtfm expands this data line and apply \_dowhichtfm. This macro selects the right result from the data line by testing with the current \_optsize value.
Optical sizes data for preloaded 8bit Latin Modern fonts:

- \texttt{fonts-resize.opm}

- \texttt{\_regtfm lmr 0 ec-lmr5 5.5 ec-lmr6 6.5 ec-lmr7 7.5 ec-lmr8 8.5 ec-lmr9 9.5}\n
- \texttt{ec-lmr10 11.1 ec-lmr12 15 ec-lmr17 *}\n
- \texttt{\_regtfm lmbx 0 ec-lmbx5 5.5 ec-lmbx6 6.5 ec-lmbx7 7.5 ec-lmbx8 8.5 ec-lmbx9 9.5}\n
- \texttt{ec-lmbx10 11.1 ec-lmbx12 *}\n
- \texttt{\_regtfm lmr 0 ec-lmr7 7.5 ec-lmr8 8.5 ec-lmr9 9.5 ec-lmr10 11.1 ec-lmr12 *}\n
- \texttt{\_regtfm lmtt 0 ec-lmtt8 8.5 ec-lmtt9 9.5 ec-lmtt10 11.1 ec-lmtt12 *}\n
- \texttt{\_setfontsize {at10pt} % default font size}\n
\section{2.13 The Font Selection System}

The basic principles of the Font Selection System used in Op\TeX{} was documented in the section 1.3.1.

\subsection{2.13.1 Terminology}

We distinguish between:

- \textit{font switchers}, they are declared by the \texttt{\font} primitive or by \texttt{\fontlet} or \texttt{\fontdef} macros, they select given font.

- \textit{variant selectors}, there are four basic variant selectors \texttt{\textit{rm}}, \texttt{\textbf{bf}}, \texttt{\textit{it}}, \texttt{\textbf{bi}}, there is a special selector \texttt{\currvar}. More variant selectors can be declared by the \texttt{\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, Op\TeX{} defines \texttt{tt} as variant selector independent of chosen font family. It selects typewriter-like font.

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

- \textit{family selectors} (for example \texttt{\Termes}, \texttt{\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 \textit{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 \textit{family context} is the current family value stored in the font context. The variant selectors declared by \texttt{\famvardef} and font modifiers declared by \texttt{\moddef} are dependent on the family context. They can have the same names but different behavior in different families.

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

\subsection{2.13.2 Font families, selecting fonts}

The \texttt{\fontfam \{Font Family\}} opens the relevant font family file where the \texttt{\{Font Family\}} is declared. The family selector is defined here by rules described in the section 2.13.1. Font modifiers and variant selectors may be declared here. The loaded family is set as current and \texttt{\textit{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 \{\langle sizespec\rangle\} command in the same sense as other font modifiers. It saves information about font size to the font context. See section 2.12. Example:

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

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

2.13.3 Math Fonts

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

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

2.13.4 Declaring font commands

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

• Font switches can be declared by \font primitive (see section 2.12.2) or by \fontlet command (see section 2.12.4) or by \fontdef command (see sections 2.13.5 and 2.13.2). 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 “build in” or declared by \moddef command. They do modifications in the font context but don’t select any font.

• “build-in” font modifiers are \setfontsize (see section 2.12), \setff (see section 2.13.9), \setfontcolor, \setletterspace and \setwordspace (see section 2.13.10). They are independent of font family.

• Font modifiers declared by \moddef depend on the font family and they are typically declared in font family files, see section 2.13.11.
- Family selectors set the given font family as current and re-set data used by the family-dependent font modifiers to initial values and to the currently used modifiers. They are declared in font family files by \_famdecl macro, see section 2.13.11.
- Font macros can be defined arbitrarily by \def primitive by users. See an example in section 2.13.8.

All declaration commands mentioned here: \font, \fontlet, \fontdef, \famvardef, \moddef, \_famdecl and \def make local assignment.

2.13.5 The \fontdef declarator in detail

The general format for \fontdef usage is

```
\fontdef⟨font switch⟩\{⟨family selector⟩ ⟨font modifiers⟩ \⟨variant selector⟩
```

where \langle family selector ⟩ and \langle font modifiers ⟩ are optional and \langle variant selector ⟩ is mandatory.

The \fontdef does the following steps. It pushes the current font context to a stack, it does modifications of the font context by given \langle family selector ⟩ and/or \langle font modifiers ⟩ and it finds the real font by \langle variant selector ⟩. This font is not selected but it is assigned to the declared \langle 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.

More about \fontdef command including examples is written in section 2.12.3.

2.13.6 The \famvardef declarator

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

```
\famvardef\{\langle new variant selector⟩ \{⟨family selector⟩ \langle font modifiers⟩ \langle variant selector⟩
```

where \langle family selector ⟩ and \langle font modifiers ⟩ are optional and \langle variant selector ⟩ is mandatory. The \langle new variant selector ⟩ should be used in the same sense as \rm, \bf etc. It can be used as the final command in next \fontdef or \famvardef declarators too. When the \langle new variant selector ⟩ is used in the normal text then it does the following steps: pushes current font context to a stack, modifies font context by declared \langle family selector ⟩ and/or \langle font modifiers ⟩, runs following \langle 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 \langle 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 “\langle var selector ⟩ is undeclared in the current family” and nothing happens. But you can declare the same variant selector by \famvardef macro in the context of a new family. Then the same command may do different work depending on the current font family.

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

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

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

A \langle family selector ⟩ can be written before \langle font modifiers ⟩ in the \famvardef parameter. Then the \langle new variant selector ⟩ is declared in the current family but it can use fonts from another family represented by the \langle 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 for Roboto family and the original meaning of \mf is lost. But OpTeX prints warning about it. There are two cases:

\begin{verbatim}
\def\mf{Metafont}
\fontfam[Roboto] % warning: "The \mf is redefined by \famvardef" is printed
\end{verbatim}

or

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

2.13.7 The \tt variant selector

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

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

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

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

Test in Termes: {\tt text}. \{\Heros\rm Test in Heros: {\tt text}}.

Test in URL \url{http://something.org}.

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

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

2.13.8 Font commands defined by \def

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

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

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

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

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

2.13.9 Modifying font features

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

The font features are part of the font context data. The macro \_setff \{[feature]\} acts like family independent font modifier and prepares a new \{feature\}. You must use a variant selector in order to reinitialize the font with the new font feature. For example \_setff{+frac}\rm or \_setff{+frac}\currvar. You can declare a new variant selector too:
Compare \( \frac{1}{2} \) or \( \frac{1}{10} \) to \( \frac{1}{2} \) or \( \frac{1}{10} \). If the used font does not support the given font feature then the font is reloaded without warning nor error, silently. The font feature is not activated.

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

### 2.13.10 Special font modifiers

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

- \texttt{\setfontsize{⟨sizespec⟩}} % sets the font size
- \texttt{\setff{⟨font feature⟩}} % adds the font feature
- \texttt{\setfontcolor{⟨color⟩}} % sets font color
- \texttt{\setletterspace{⟨number⟩}} % sets letter spacing
- \texttt{\setwordspace{⟨scaling⟩}} % modifies word spacing

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

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

- These colors of fonts are implemented using LuaTeX internal font feature. This is different approach than using colors in section 2.20.

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

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

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

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

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

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

### 2.13.11 How to create the font family file

The font family file declares the font family for selecting fonts from this family at the arbitrary size and with various shapes. Unicode fonts (OTF) are preferred. The following example declares the Heros family:
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\} \{comments about math fonts\}
\{font-for-testing\}
\def\fontnamegen{\{font name or font file name generated\}}

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

- 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 \ifmathloading 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 \{selector\} \\fontnamegen \sizespec
Note that the extended \font syntax \font\{\font\{\selector\}\{\font\{\name\}\{\features\}\}\{\size\}\} or \font\{\selector\}{\font\{\font\file\}\{\features\}\}\{\size\}\} is expected here.

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

\moddef\resetmod{\_fvars Regular Bold Italic BoldItalic }

and define the \fontnamegen in the last parameter of the \famdecl by:

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

The following auxiliary macros are used here:

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

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

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

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

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

\fsetV \{word-a\}={\{value-a\}},\{word-b\}={\{value-b\}} ...etc. terminated by a space

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

Example 3
If both font modifiers \caps, \cond were applied in Heros family, then \def\_capsV{+smcp;\_ffonum;} and \def\_condV{cn} were processed by these font modifiers. If a user needs the \bf variant at 11pt now then the

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

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

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

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

The \moddef macro
has the syntax \moddef\{\modifier\}\{\what\}. 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 \texttt{famvardef} macro has the same features.

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

\begin{verbatim}
\protected\def\langle Familyselector \rangle{%
  \_def\_currfamily \langle Familyselector \rangle% 
  \_def\_fontnamegen {...}% this is copied from 7-th parameter of \_famdecl
  \resetmod
  \_initfontfamily
}\end{verbatim}

The \texttt{\_initfontfamily} must be run after modifier’s declaration. It runs the \texttt{\langle Familyselector \rangle} and it runs \texttt{\_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 \texttt{\langle 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 OpTEX warns about it. See the end of section 2.13.4.

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

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

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

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

The family selector first resets all modifiers data by \texttt{\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, \texttt{\Termes} does \texttt{\resetmod} followed by \texttt{\caps\cond}. The \texttt{\caps} is applied and \texttt{\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 \texttt{\_wordM}. You can be sure that such a name does not influence the private namespace used by OpTEX.

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

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

If you need to create a font family file with a non-Unicode font, you can do it. The \texttt{\_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 \texttt{\_optname} macro when \texttt{\_fontnamegen} in expanded. This macro is fully expandable and its input is \texttt{\langle internal-template \rangle} and its output is a part of the font file name \texttt{\langle size-dependent-template \rangle} with respect to given optical size.

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

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

In general:

\begin{verbatim}
\_regoptsizes \langle internal-template \rangle \langle general-output-template \rangle \langle resizing-data \rangle
\end{verbatim}

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

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

We are using a special macro \LRegfont in f-lmfonts.opm. It sets the file names to lowercase and enables us to use shortcuts instead of real \resizingdata. There are shortcuts \RegoptFS, \RegoptT, etc. here. The collection of \internaltemplates \ are declared, each of them covers a collection of real file names.

The \fontalias{} \internaltemplate \ declares \newtemplate \ with the same meaning as previously declared \internaltemplate.

The \optname \ macro can be used even if no optical sizes are provided by a font family. Suppose that font file names are much more chaotic (because artists are very creative people), so you need to declare more systematic \internaltemplates \ and do an alias from each \internaltemplate \ to \realname. For example, you can do it as follows:

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

Another example of a font family with optical sizes is Antykwa Półtawskiego. The optical sizes feature is deactivated by default and it is switched on by \osize \ font modifier:
2.13.13 How to register the font family in the Font Selection System

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

You can use your font file in sense of the previous paragraph without registering it. But problem is that such families are not listed when \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: \texttt{fams-ini.opm} and \texttt{fams-local.opm}. The second file is optional. Users can create it and write to it the information about user-defined families using the same syntax as in existed file \texttt{fams-ini.opm}.

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

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

```
\_faminfo [Catalogue] {Catalogue of all registered font families} {fonts-catalog} {} \\
\_famalias [Catalog] \\
\_faminfo [Computer Modern like family:] \\
\_famfrom {GUST} \\
\_faminfo [Latin Modern] {TeX Gyre fonts based on Coputer Modern} {f-lmfonts} \\
\ttprop,\ttprop\bolder,\quotset: {\rm\bf\it\bi} \\
\caps: {\rm\bf\it\bi} \\
\ttlight,\ttcond,\dunhill: {\rm\bf\it\bi} \\
\_famalias [Latin Modern Fonts] \_famalias [lm]
```

... etc.

The \_faminfo command has the syntax:

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

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

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

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

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

73
2.13.14 Notices about extension of \font primitive

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

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

The OpTeX format is initialized by \luatex engine by default but you can initialize it by \luahbtex engine too. Then the harfbuzz library is ready to use for font rendering as an alternative to build-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 \luahbtex engine is not used then \mode=harf is ignored. See Luaotfload documentation for more information.

2.13.15 Implementation of the Font Selection System

\initunifonts macro extends LuaTeX’s font capabilities, in order to be able to load Unicode fonts. Unfortunately, this part of OpTeX depends on luaotfload package, which adapts ConTeXt’s generic font loader for plain \TeX and \LaTeX.\luaotfload uses Lua functions from \LaTeX’s \luatexbase namespace, we provide our own replacements. Moreover, \initunifonts switches with the \doresizefont macro to OTF mode which is represented by the macro \doresizeunifont. This mode includes a fallback to TFM mode if \fontnamegen is not defined. Finally, \initunifonts sets itself to relax because we don’t want to do this work twice.

\ttunifont is default font for \tt variant if f-lmfonts.om (or another font file where \tt is declared) is not loaded.

The \famdecl command defines font family. This family is loaded by \fontfam macro. The \fontfam macro needs it. 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 actual meaning of the \fontnamegen macro. The \mathfaminfo is saved for usage in the catalog.
\def\famdecl [#1]#2#3#4#5#6#7#8{% 
\unless\ifcsname \string#2:main.fam\endcsname 
\ifnum#1=7 \errmessage{Family [#1] skipped, font "#7" not found}\else 
\edef\currfamily \csname \string#2\endcsname
\edef\mathfaminfo{#6} 
\wterm {FONT: [#1] -- \csname#2\endcsname (#3) mods:{#4} vars:{#5} math:{#6}} 
\unless \ifx #2\undefined 
\opwarning{#2 is redefined by \string\famdecl\space [#1]} 
\fi 
\protectededef#2{\def\noexpand\currfamily {\csstring #2} 
\def\mathfaminfo{#6} 
\wterm {FONT: [#1] -- \csname#2\endcsname (#3) mods:{#4} vars:{#5} math:{#6}}} 
\fi 
\else 
\csname \string#2:main.fam\endcsname \reloading \rm \let \csname \string#2\endcsname \fi 
} 
\def\initfontfamily{% 
\csname \string#2:main.fam\endcsname \reloading \rm 
} 
\def\regoptsizes {⟨internal-template⟩ ⟨left-output⟩ ⟨resizing-data⟩} 
\regoptsizes 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⟩ ⟨resizing-data⟩ in the loop. 
\def\optfontalias {⟨template A⟩}{⟨template B⟩} is defined as \let\oz:⟨templateA⟩ =\oz:⟨templateB⟩. 
\def\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. 
\def\fsetV {key}=⟨value⟩, ..., ⟨key⟩=⟨value⟩ expands to \def\{⟨key⟩\}=⟨value⟩ in the loop. 
\onlyif ⟨key⟩=⟨value-a⟩, ⟨value-b⟩, ..., ⟨value-z⟩: ⟨what⟩ runs ⟨what⟩ only if the \{⟨key⟩\} is defined as ⟨value-a⟩ or ⟨value-b⟩ or ... or ⟨value-z⟩. 
\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.
The \moddef \{\langle modifier \rangle \}{\{\langle data \rangle \}} simply speaking does \def\langle modifier \rangle {\{\langle data \rangle \}}, but we need to re-
respect the family context. In fact, \protected\def f:\langle current family \rangle \{\langle modifier \rangle \}{\{\langle data \rangle \}} is performed and the \langle modifier \rangle is defined as \famdepend\langle modifier \rangle {\{f:\langle current family \rangle }{\langle data \rangle \}}. It expands to \f:\langle current family \rangle \{\langle modifier \rangle \} value if it is defined or it prints the warning. When the \langle current family \rangle \{\langle modifier \rangle \} value is changed then we can declare the same \langle modifier \rangle with a different meaning.

When a user declares a prefixed variant of the \langle modifier \rangle then unprefixed modifier name is used in internal macros, this is the reason why we are using the \remifirstunderscore\_tmp (where \_tmp expands to \langle something \rangle or to \langle something \rangle). The \remifirstunderscore redefines \_tmp in the way that it expands only to \langle something \rangle without the first _.

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

\addtomodlist \{\langle font modifier \rangle \} 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.

The \famdepend \{\langle XX \rangle \}{\{\langle data \rangle \}} uses analogical trick like \moddef with the \famdepend macro. The auxiliary \famdepend \{\langle XX \rangle \}{\langle tryload\langle XX \rangle \} \{\langle data \rangle \} \} is used:

- \def \tryload\{\langle currfam \rangle \}{\langle XX \rangle } \{\fontdef \ten\langle XX \rangle \{\langle data \rangle \} \}\} loads font \ten\langle XX \rangle ,
- \protected\def f:\langle XX \rangle \{\famdepend \langle XX \rangle \} \{f:\langle currfam \rangle \{\langle XX \rangle \} \}\}.
- \def \_f:\langle currfam \rangle \{\langle XX \rangle \} \{\tryload\{\langle currfam \rangle \}{\langle XX \rangle } \{\langle data \rangle \} \}\} keeps family dependent definition,
- \def \_currvar\_ten\langle XX \rangle \{\langle XX \rangle \}\} in order to the \currvar macro work correctly.

\famdependtt behaves somewhat differently: it doesn’t re-define the \ttt macro which is defined as \tryloadtt \{\langle ten\tt \}\} in sections 2.14 and 2.16.2. It only re-defines the internal \tryloadtt macro.
The `\fontfam` does:

- Convert its parameter to lower case and without spaces, e.g. `{fontfamily}`.
- If the file `f-{fontfamily}.opm` exists read it and finish.
- Try to load user defined `fams-local.opm`.
- If the `{fontfamily}` is declared in `fams-local.opm` or `fams-ini.opm` read relevant file and finish.
- Print the list of declared families.

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

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

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

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

\_famalias does nothing or prints the text on the terminal.
\famfrom declares type foundry or owner or designer of the font family. It can be used in \fams-init.opm or \fams-local.opm and it is printed in the font catalog.

\input \fams-init.opm
\let \famfile=\undefined
\famfrom={}

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

\input \fams-init.opm
\let \famfile=\undefined
\famfrom={}

The font features are managed in the \fontfeatures macro. They have their implicit values saved in the \defaultfontfeatures and the \setff \{\features\} can add next font features. If there is the same font feature as the newly added one then the old value is removed from the \fontfeatures list.

\input \fams-init.opm
\let \famfile=\undefined
\famfrom={}

The \setfontcolor and \setletterspace are macros based on the special font features provided by LuaTeX (and by \XeTeX too but it is not our business). The \setwordspace recalculates the \fontdimen2,3,4 of the font using the \setwsp macro which is used by the \doesizeunifont macro. It activates a dummy font feature +Ws too in order the font is reloded by the \font primitive (with independent \fontdimen registers).
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 \texttt{\setmathsizes [\langle text-size\rangle/\langle script-size\rangle/\langle scriptscript-size\rangle]} and then load all math fonts in given sizes by \texttt{\normalmath} or \texttt{\boldmath} macros. For example

\begin{verbatim}
\setmathsizes[12/8.4/6]\normalmath ... math typesetting at 12 pt is ready.
\end{verbatim}

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

The optical sizes of Computer Modern fonts, AMS, and other fonts are declared here.
\_loadmathfamily \langle number \rangle \langle font \rangle loads one math family, i.e. the triple of fonts in the text size, script size and script-script size. The \langle font \rangle is \langle font-id \rangle used in the \_regtfm parameter or the real TFM name. The \textfamily is saved as \textfamily\langle number \rangle.

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

\_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).

\_def \_corrmsize#1 \{ \_ptmunit=#1 \_ptunit \} % for corrections of sizes in diferent fonts

\_def \_loadmathfamily \#1 \#2 \{ %
  \_edef \_optsizesave{\_the \_optsize} %
  \_optsize=\_sizem\text \_font\mF=\_whichtfm{\#2} \_textfont\#1=\_mF %
  \_optsize=\_sizem\script \_font\mF=\_whichtfm{\#2} \_scriptfont\#1=\_mF %
  \_optsize=\_sizem\sscript \_font\mF=\_whichtfm{\#2} \_scriptscriptfont\#1=\_mF %
  \_optsize=\_optsizesave \_ptmunit=\_ptunit %
\}

\_def \_setmathfamily \#1 \#2 \{ %
  \_let \_mF=\#2 \%
  \_edef \_optsizesave{\_the \_optsize} %
  \_optsize=\_sizem\text \_let\#2=\#2 \_textfont\#1=\#2 %
  \_optsize=\_sizem\script \_let\#2=\#2 \_scriptfont\#1=\#2 %
  \_optsize=\_sizem\sscript \_let\#2=\#2 \_scriptscriptfont\#1=\#2 %
  \_optsize=\_optsizesave \_ptmunit=\_ptunit \_let\#2=\_mF %
\}

\_def \_setmathsizes[\#1/\#2/\#3] \{ \_ptmunit=\_ptunit %
  \_def \_sizem\text=\#1 \_ptmunit \%
  \_def \_sizem\script=\#2 \_ptmunit \%
  \_def \_sizem\sscript=\#3 \_ptmunit \%
\}

\_newdimen \_ptunit \_ptunit=1\ptunit

\_public \_setmathsizes \_ptunit \_ptunit ;

The \_setmathsizes 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.

\_def \_setmathdimens\% Plain\TeX sets these dimens for 10pt size only:
  \_delimitershortfall=0.8 \_fontdimen6\_textfont3
  \_nulldelimiterspace=0.12 \_fontdimen6\_textfont3
  \_scriptspace=0.05 \_fontdimen6\_textfont3
  \_skewchar\_textfont1=127 \_skewchar\_scriptfont1=127
  \_skewchar\_scriptscriptfont1=127
  \_skewchar\_textfont2=48 \_skewchar\_scriptfont2=48
  \_skewchar\_scriptscriptfont2=48
  \_skewchar\_textfont6=127 \_skewchar\_scriptfont6=127
  \_skewchar\_scriptscriptfont6=127
\}

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

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

There is a problem: The \(x_n\) is tokenized as \(x\,\_\,n\) and it works without problems. But \(int_a^b\) is tokenized as \(\int\,a^b\). The control sequence \(\int\) is defined. We must write \(\int\,a^b\).

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

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

All mathcodes are set to equal values as in plain\TeX. But all encoding-dependent declarations (like these) will be set to different values when a Unicode-math font is used.
All control sequences declared by \texttt{\mathchardef} are supposed (by default) only for public usage. It means that they are declared without _ prefix. If such sequences are used in internal \texttt{Op\TeX} macro then their internal prefixed form is declared using \texttt{\private} macro.

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

The \texttt{math functions like log, sin, cos are declared in the same way as in \texttt{plain\TeX}, but they are \texttt{\protected} in \texttt{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.

\big, \Big, \bigg, \Bigg, \Bigl, \Bigm, \Bigr, \biggl, \biggm, \biggr, \Biggl, \Biggm, \Bigg are based on the \scalebig macro because we need the dependency on the various sizes of the fonts.
Math relations defined by the \texttt{\jointrel} plain \TeX{} macro:

\begin{verbatim}
\protected\def\_jointrel{\_mathrel{\_smash-}} % \_smash, because - has the same height as +
\protected\def\\relbar{\_mathrel{=}}
\mathchardef\lhook=\312C
\protected\def\\hookrightarrow{\_lhook\_joinrel\rightarrow}
\mathchardef\rhook=\312D
\protected\def\\hookleftarrow{\_leftarrow\_joinrel\rhook}
\protected\def\\bowtie{\_mathrel\triangleright\_joinrel\mathrel\triangleleft}
\protected\def\\models{\_mathrel|\_joinrel=}\protect\def\\Longrightarrow{\_Relbar\_joinrel\Rightarrow}
\protected\def\\longrightarrow{\_relbar\_joinrel\rightarrow}
\protected\def\\longleftarrow{\_leftarrow\_joinrel\relbar}
\protected\def\\Longleftarrow{\_Leftarrow\_joinrel\Relbar}
\protected\def\\longmapsto{\_mapstochar\longrightarrow}
\protected\def\\longleftrightarrow{\_leftarrow\_joinrel\rightarrow}
\protected\def\\Longleftrightarrow{\_Leftarrow\_joinrel\Rightarrow}
\protected\def\\iff{\_thicksk\_\Longleftrightarrow\_thicksk}
\end{verbatim}

\ldots, \cdots, \vdots, \ddots from plain \TeX{}

\begin{verbatim}
\mathchardef\ldotp=\613A % ldotp as a punctuation mark
\mathchardef\cdotp=\6201 % cdot as a punctuation mark
\mathchardef\colon=\603A % colon as a punctuation mark
\public \ldotp \cdotp \colon
\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.7em\_vbox{\_kern.7em\_hbox{.}}\_mkern1mu}}
\end{verbatim}

\ldots \cdots \vdots \ddots ;

\adots inspired by plain \TeX{}

\begin{verbatim}
\protected\def\_adots{\_mathinner{\_mkern1mu\_raise.7em\_vbox{\_kern.7em\_hbox{.}}\_mkern2mu\_raise.4em\_hbox{.}\_mkern2mu\_raise.7em\_vbox{\_kern.7em\_hbox{.}}\_mkern1mu}}
\end{verbatim}

\adots ;

Math accents (encoding dependent declarations).

\begin{verbatim}
\protected\def\acute{\_mathaccent\7013}
\protected\def\grave{\_mathaccent\7012}
\protected\def\ddot{\_mathaccent\707F}
\protected\def\tilde{\_mathaccent\707E}
\end{verbatim}

\texttt{\_public \ldots \adots \\ldots \adots ;
\mathaccent\7013 \mathaccent\7012 \mathaccent\707F \mathaccent\707E ;
\_public \ldots \mathaccent\7016}
Macros based on \delimiter, \widetilde, and \hspace primitives.
The \texttt{\cases} and \texttt{\bordermatrix} macros are almost identical as in plain \TeX. You can simply re-define \texttt{\bordermatrix} with other delimiters using the common \texttt{\bordermatrixwithdelims} macro.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\not< becomes \nless
\not> becomes \ngtr
if \notXXX is defined, \not\XXX becomes \notXXX;
if \nXXX is defined, \not\XXX becomes \nXXX;
otherwise, \notXXX is done in the usual way.

The ⟨math list⟩ behaves like ⟨math list⟩, but you can use the following commands in the ⟨math list⟩:

\mathtextstyles{⟨math list⟩}
• \texttt{\currstyle} which expands to \texttt{\displaystyle, \textstyle, \scriptstyle} or \texttt{\scriptscriptstyle} depending on the current math style when \texttt{\mathstyles} was opened.

• \texttt{\dobystyle{⟨D⟩}{⟨T⟩}{⟨S⟩}{⟨SS⟩}} is expandable macro. It expands to \texttt{⟨D⟩, ⟨T⟩, ⟨S⟩ or ⟨SS⟩} depending on the current math style when \texttt{\mathstyles} was opened.

• The value of the \texttt{\stylenum} is 0, 1, 2 or 3 depending on the current math style when \texttt{\mathstyles} was opened.

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

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

\texttt{\def\mathframe#1{\mathstyles{\frame{$\currstyle\cramped #1$}}}}.

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

\texttt{$\cramped{x^2\over y^2}$}.

The \texttt{\mathbox{⟨text⟩}} macro is copied from OPmac trick 078. It behaves like \texttt{\hbox{⟨text⟩}} but the \texttt{⟨text⟩} is scaled to a smaller size if it is used in scriptstyle or scriptscript style.

The \texttt{\_textmff} and \texttt{\_scriptmff} are redefined in order to respect optical sizes. If we are in script style then the math mode starts in text style, but optical size is given to script style. The \texttt{\mathbox} in non-Unicode math respects optical sizes using different principle.

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

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

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

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

The \texttt{\loadmath} macro was sucessfully tested on:

\texttt{\loadmath{[XITSMath-Regular]} \ldots XITS MATH}
\texttt{\loadmath{[latinmodern-math]} \ldots Latin Modern Math}
\texttt{\loadmath{[texgyretermes-math]} \ldots TeXGyre Termes Math}
\texttt{\loadmath{[texgyrebonum-math]} \ldots TeXGyre Bonum Math}
2.16.1 Unicode-math macros preloaded in the format

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

- define \unimathfont as ⟨Unicode-math font⟩,
- 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{⟨bold-font⟩} \to {⟨normal-font⟩} defines \unimathboldfont as ⟨bold-font⟩ only if \unimathfont is defined as ⟨normal-font⟩. It is used when \boldmath macro is run. When no \unimathboldfont is defined then the \boldmath macro use “fake bold” generated by embolden LuaTeX font feature.

The Unicode version of the \normalmath and \boldmath macros are defined here as \normalunimath and \boldunimath macros. They are using \_setunimathdimens in a similar sense as \_setmathdimens. You can combine more fonts if you register them to another math families (5, 6, 7, etc.) in the \normalmath macro.
The default value of \_normalunimath shows a combination of base Unicode-math font with 8bit Math font at family 4. See definition of \script macro where \_fam4 is used.

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

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

\loadumathfamily \langle number \rangle \langle font \rangle \langle font features \rangle loadsthe given Unicode-math fonts in three sizes given by the \setmathsizes macro and sets it as the math family \langle number \rangle. The \langle font features \rangle are added to the default \_mfontfeatures and to the size-dependent features +ssty=0 if script size is asked or +ssty=1 if scriptscriptsize is asked. If the math family 1 is loaded then the family 2 and 3 are set by the same font because \LaTeX needs to read dimension information about generating math formulae from these three math families. All information needed by \LaTeX is collected in single Unicode-math font. The \corrmsize \langle factor \rangle \langle space \rangle can be used just before \loadumathfamily, see section 2.14 for more information.

The \_textmff, \_scriptmff and \_sscriptmff are additional font features for text, script and scriptscript sizes respectively. They are locally re-defined in \textbox macro.
Unicode math font includes all typical math alphabets together, user needs not to load more \TeX math families. These math alphabets are encoded by different parts of Unicode table. We need auxiliary macros for setting mathcodes by selected math alphabet.  

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

\_umathrangegreek \_first is the same as \_umathrange {⟨alpha⟩-⟨omega⟩}\_first. \_umathrangegreek \_first is the same as \_umathrange {⟨Alpha⟩-⟨Omega⟩}\_first. \_greekdef \_relax defines each control sequence as a normal character with codes \_umathnumB, \_umathnumB+1, \_umathnumB+2 etc. It is used for redefining the control sequences for math Greek \texttt{\textbackslash alpha}, \texttt{\textbackslash beta}, \texttt{\textbackslash gamma} etc.

2.16.2 Macros and codes set when \texttt{\textbackslash loadmatfont} is processed

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

The control sequences for \texttt{\textbackslash alpha}, \texttt{\textbackslash beta} etc are redefined here. The \texttt{\textbackslash alpha} expands to the character with Unicode ^*03B1*, this is a normal character α. You can type it directly in your editor if you know how to do this.
The math alphabets are declared here using the \_umathrange\{range\}\{class\}\{family\}\{starting-code\} macro.

\_protected\_def\_rmvariables \{ \_umathrange{A-Z}\_ncharrmA \_umathrange{a-z}\_ncharrma \}
\_protected\_def\_bfvariables \{ \_umathrange{A-Z}\_ncharbfA \_umathrange{a-z}\_ncharbfa \}
\_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\_bcalvariables \{ \_umathrange{A-Z}\_ncharbcA \_umathrange{a-z}\_ncharbca \}
\_protected\_def\_frakvariables \{ \_umathrange{A-Z}\_ncharfrA \_umathrange{a-z}\_ncharfra \}
\_protected\_def\_bfrakvariables \{ \_umathrange{A-Z}\_ncharbrA \_umathrange{a-z}\_ncharbra \}
\_protected\_def\_bbvariables \{ \_umathrange{A-Z}\_ncharbbA \_umathrange{a-z}\_ncharbba \}
\_protected\_def\_sansvariables \{ \_umathrange{A-Z}\_ncharsnA \_umathrange{a-z}\_ncharsna \}
\_protected\_def\_bsansvariables \{ \_umathrange{A-Z}\_ncharbsA \_umathrange{a-z}\_ncharbsa \}
\_protected\_def\_isansvariables \{ \_umathrange{A-Z}\_ncharsiA \_umathrange{a-z}\_ncharsia \}
\_protected\_def\_bisansvariables \{ \_umathrange{A-Z}\_ncharsxA \_umathrange{a-z}\_ncharsxa \}
\_protected\_def\_ttvariables \{ \_umathrange{A-Z}\_ncharttA \_umathrange{a-z}\_nchartta \}

\_protected\_def\_itgreek \{ \_umathrangegreek71\_greekita \}
\_protected\_def\_rmgreek \{ \_umathrangegreek71\_greekrma \}
\_protected\_def\_bfgreek \{ \_umathrangegreek71\_greekbfa \}
\_protected\_def\_bigreek \{ \_umathrangegreek71\_greekbia \}
\_protected\_def\_bsansgreek \{ \_umathrangegreek71\_greeksna \}
\_protected\_def\_bisansgreek \{ \_umathrangegreek71\_greeksia \}
\_protected\_def\_itGreek \{ \_umathrangeGREEK71\_greekitA \_setnablait \}
\_protected\_def\_rmGreek \{ \_umathrangeGREEK71\_greekrmA \_setnablarm \}
\_protected\_def\_bfGreek \{ \_umathrangeGREEK71\_greekbfa \_setnablaf \}
\_protected\_def\_biGreek \{ \_umathrangeGREEK71\_greekbia \_setnablabi \}
\_protected\_def\_bsansGreek \{ \_umathrangeGREEK71\_greeksna \_setnablabsans \}
\_protected\_def\_bisansGreek \{ \_umathrangeGREEK71\_greeksia \_setnablabisans \}

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

\_def \_setnabla \{ \_Umathcode"2207 = 7 1} 
\_def \_setnablalbf \{ \_setnabla"02207 \}
\_def \_setnablait \{ \_setnabla"03B1 \}
\_def \_setnablabi \{ \_setnabla"03BC \}
\_def \_setnablalbsans \{ \_setnabla"1D756 \}
\_def \_setnablabisans \{ \_setnabla"1D7AA \}

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

\begin{verbatim}
\protected\def\bf
\{\tryloadbf\tenbf \inmath\{\bfvariables\bfgreek\bfGreek\bfdigits\}\}
\protected\def\bi
\{\tryloadbi\tenbi \inmath\{\bivariables\bigreek\bfGreek\bfdigits\}\}
\public \bf \bi \\
\inmath \{\cmds\} \applies \cmds \only \in \texttt{math mode}.
\end{verbatim}

\begin{verbatim}
\protected\def\imath\{\relax\ifmmode\fi\}
% to keep off \loop processing in text mode
\end{verbatim}

% You can redefine these macros to follow your wishes.
% For example, you need upright lowercase greek letters, you don’t need
% \texttt{\bf} and \texttt{\bi} behave as sans serif in math, ...
\protected\def\rm
\{\tryloadrm\tenrm \inmath\{\rmdigits\}\}
\protected\def\it
\{\tryloadit\tenit \inmath\{\itgreek\rmdigits\}\}
\protected\def\bf
\{\tryloadbf\tenbf \inmath\{\bsansvariables\bsansgreek\bsansGreek\bsansdigits\}\}
\protected\def\bi
\{\tryloadbi\tenbi \inmath\{\bsansvariables\bsansgreek\bsansGreek\bsansdigits\}\}
\protected\def\tt
\{\tryloaddt\tentt \inmath\{\ttvariables\ttdigits\}\}
\protected\def\bbchar
\{\bbvariables\bbdigits\}
\protected\def\cal
\{\calvariables\}
\protected\def\frak
\{\frakvariables\}
\protected\def\sans
\{\sansvariables\sansdigits\}
\protected\def\script
\{\rvariables\rmdigits\itgreek\rmGreek\}
\public \rm \it \bf \bi \tt \bbchar \cal \frak \sans \script \mit \\
\end{verbatim}

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

\begin{verbatim}
002E;P
002F;B
0030..0039;N
003A;P
003B;B
003C;R
003D;R
003E;N
003F;P
0040;N
0041..005A;A
005B;D
005C;B
005D;C
005E;N
005F;N
\end{verbatim}

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

95
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:

```
\begin{verbatim}
70 \UnicodeMathSymbol{00393}{\mupGamma }{\mathalpha}{capital gamma, greek}\
71 \UnicodeMathSymbol{00394}{\mupDelta }{\mathalpha}{capital delta, greek}\
72 \UnicodeMathSymbol{00395}{\mupEpsilon }{\mathalpha}{capital epsilon, greek}\
73 \UnicodeMathSymbol{00396}{\mupZeta }{\mathalpha}{capital zeta, greek}\
74 \UnicodeMathSymbol{00397}{\mupEta }{\mathalpha}{capital eta, greek}\
75 \UnicodeMathSymbol{00399}{\mupTheta }{\mathalpha}{capital theta, greek}\
76 \UnicodeMathSymbol{0039A}{\mupIota }{\mathalpha}{capital iota, greek}\
77 \UnicodeMathSymbol{0039B}{\mupLambda }{\mathalpha}{capital lambda, greek}\
78 \UnicodeMathSymbol{0039C}{\mupMu }{\mathalpha}{capital mu, greek}\
79 \UnicodeMathSymbol{0039D}{\mupNu }{\mathalpha}{capital nu, greek}\
80 \UnicodeMathSymbol{0039E}{\mupXi }{\mathalpha}{capital xi, greek}\
81 \UnicodeMathSymbol{0039F}{\mupOmicron }{\mathalpha}{capital omicron, greek}\
82 \UnicodeMathSymbol{003A0}{\mupPi }{\mathalpha}{capital pi, greek}\
83 \UnicodeMathSymbol{003A1}{\mupRho }{\mathalpha}{capital rho, greek}\
84 \UnicodeMathSymbol{003A3}{\mupSigma }{\mathalpha}{capital sigma, greek}\
85 \end{verbatim}
```

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

```
\begin{verbatim}
183 \begin{verbatim}
184 \def\UnicodeMathSymbol#1#2#3#4{% 
185 \ifnum#1=\Umathcodenum#1 % the code isn’t set by mathclass.opm 
186 \Umathchardef#2=0 1 #1 \Umathcode#1=0 1 #1 
187 \else \Umathcharnumdef#2=\Umathcodenum#1 \fi 
188 \ifx#3\mathopen \def#2{\Udelimiter 4 1 #1} \fi 
189 \ifx#3\mathclose \def#2{\Udelimiter 5 1 #1} \fi 
190 \ifx#3\mathaccent \def#2{\Umathaccent fixed 7 1 #1} \fi 
191 \end{verbatim}
\end{verbatim}
```

Many special characters must be declared with care...

```
\begin{verbatim}
219 \global\Udelcode`<1 "027E8 % these characters have different meaning 
220 \global\Udelcode`>1 "027E9 % as normal and as delimiter 
221 \mit % default math alphabets setting 
222 \% hyphen character is transformed to minus: 
223 \Umathcode `{~ = 2 \*2112 
224 \% mathclass defines : as Punct, plain.tex as Rel, we keep mathclass, 
225 \% i.e. there is difference from plain.tex, you can use $f:A\to B$. 
226 \% mathclass defines ! as Ord, plain.tex as Close 
227 \Umathcode `' = 5 1` % keep plain.tex declaration 
228 \% \mathchardef \mathexclam = 5 1` % keep plain.tex declaration 
229 \% \mathchardef \mathexclam = 5 1` % keep plain.tex declaration 
230 \% \mathchardef \mathexclam = 5 1` % keep plain.tex declaration 
\end{verbatim}
```

96
Aliases are declared here. They are names not mentioned in the \texttt{unimath-table.opm} file but commonly used in \TeX.

\begin{verbatim}
290 \_let \setminus=\smallsetminus
291 \_let \diamond=\smwhtdiamond
292 \_let \colon=\mathcolon
293 \_let \bullet=\smblkcircle
294 \_let \circ=\vysmwhtcircle
295 \_let \bigcirc=\mdlgwhtcircle
296 \_let \to=\rightarrow
297 \_let \le=\leq
298 \_let \ge=\geq
299 \_let \neq=\ne
300 \_protected\_def \triangle {\mathord{\bigtriangleup}}
301 \_let \emptyset=\varnothing
302 \_let \hbar=\hslash
303 \_let \land=\wedge
304 \_let \lor=\vee
305 \_let \owns=\ni
306 \_let \gets=\leftarrow
307 \_let \mathring=\ocirc
308 \_let \lnot=\neg
309 \_let \longdivisionsign=\longdivision
310 \_let \backepsilon=\upbackepsilon
311 \_let \eth=\matheth
312 \_let \dbkarow=\dbkarrow
313 \_let \hksearow=\hksearrow
314 \_let \hkswarow=\hkswarrow
315 \_let \upalpha=\mupalpha
316 \_let \upbeta=\mupbeta
317 \_let \upgamma=\mupgamma
318 \_let \updelta=\mupdelta
319 \_let \upepsilon=\mupvarepsilon
320 \_let \upvarrho=\mupvarrho
321 \_let \upvarsigma=\mupvarsigma
322 \_let \upsigma=\mupsigma
323 \_let \upomega=\mupomega
324 \_let \upvartheta=\mupvartheta
325 \_let \upphi=\mupphi
326 \_let \upchi=\mupchi
327 \_let \uppsi=\muppsi
328 \_let \upvarpi=\mupvarpi
\end{verbatim}

The \texttt{\_not} macro is redefined here. If the \texttt{\_not\langle char\rangle} is defined (by \texttt{\_negationof}) then this macro is used. Else centered / is printed over the \langle char\rangle.

\begin{verbatim}
354 \_protected\_def \_not\#1\{
355 \_tryce{\_not\not\_cstring\#1}{\mathrel{\mathstyles{\_setbox0=\_hbox{\_math\_currstyle\#1}}}}
356 \_hbox to\_wd0{\_hss\_currstyle/\_hss}\_kern-\_wd0\_box0
357 \}}
\end{verbatim}

98
\def\negationof #1#2{\_ea\_let \_csname _not!\_csstring#1\endcsname =#2}

\negationof = \neq
\negationof < \nless
\negationof > \ngtr
\negationof \gets \nleftarrow
\negationof \simeq \nsime
\negationof \equal \ne
\negationof \le \nleq
\negationof \ge \ngeq
\negationof \greater \ngtr
\negationof \forksnot \forks
\negationof \in \notin
\negationof \mid \nmid
\negationof \cong \ncong
\negationof \leftarrow \nleftarrow
\negationof \rightarrow \nrightarrow
\negationof \leftrightarrow \nleftrightarrow
\negationof \Leftarrow \nLeftarrow
\negationof \Leftrightarrow \nLeftrightarrow
\negationof \Rightarrow \nRightarrow
\negationof \exists \nexists
\negationof \ni \nni
\negationof \paralel \nparalel
\negationof \sim \nsim
\negationof \approx \napprox
\negationof \equiv \nequiv
\negationof \asymp \nasymp
\negationof \lesssim \nlesssim
\negationof \ngtrsim \ngtrsim
\negationof \lessgtr \nlessgtr
\negationof \gtrless \ngtrless
\negationof \prec \nprec
\negationof \succ \nsucc
\negationof \subset \nsubset
\negationof \supset \nsupset
\negationof \subseteq \nsubseteq
\negationof \supseteq \nsupseteq
\negationof \vdash \nvdash
\negationof \vDash \nvDash
\negationof \Vdash \nVdash
\negationof \VDash \nVDash
\negationof \preccurlyeq \npreccurlyeq
\negationof \succcurlyeq \nsucccurlyeq
\negationof \sqsubseteq \nsqsubseteq
\negationof \sqsupseteq \nsqsupseteq
\negationof \vartriangleleft \nvartriangleleft
\negationof \vartriangleright \nvartriangleright
\negationof \trianglelefteq \ntrianglelefteq
\negationof \trianglerighteq \ntrianglerighteq
\negationof \vinfty \nvinfty
\negationof \unimath-codes.opm

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. See http://tex.stackexchange.com/questions/308749 for technical details about Unicode-math.
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 \texttt{\fontfam[termes]} or by \texttt{\loadmath{⟨math-font⟩}} and then you can do \texttt{\input print-unimath.opm}. The big table with all math symbols is printed.

\begin{verbatim}
\_codedecl \_undefined {Printing Unicode-math table \string<2020-06-08>}
\begingroup
\_def\UnicodeMathSymbol#1#2#3#4{% 
  \_ifnum#1>"10000 \_endinput \_else \_printmathsymbol{#1}{#2}{#3}{#4}\_fi}
\_def\UnicodeMathSymbolA#1#2#3#4{% 
  \_ifnum#1>"10000 \_printmathsymbol{#1}{#2}{#3}{#4}\_fi}
\_def\_printmathsymbol#1#2#3#4{\hbox{\hbox to2em{$#2{}$} \hss} \hbox to3em {\small\printop#3 \hss} \tt\string#2\_trycs{_eq:\string#2}{}}
\_def\_eq#1#2{\sdef{_eq:\string#2}{=\string#1}}
\_eq \diamond \smwhtdiamond \_eq \bullet \smblkcircle \_eq \circ \vysmwhtcircle
\_eq \bigcirc \mdlgwhtcircle \_eq \to \rightarrow \_eq \le \leq \_eq \ge \geq \_eq \neq \ne\emptyset \_eq \hbar \hslash \_eq \land \wedge \_eq \lor \vee \_eq \owns \ni \_eq \gets \leftarrow \_eq \mathring \ocirc \_eq \lnot \neg \_eq \backepsilon \upbackepsilon \_eq \eth \matheth \_eq \dbkarow \dbkarrow \_eq \drbkarow \drbkarrow \_eq \hksearrow \hksearrow \_eq \hksvarrow \hksvarrow
\_tracinglostchars=0
\fontdef\small{\setfontsize{at5pt}\rm}
\_def\mathop{Op}
\_def\mathalpha{Alph}
\_def\mathord{Ord}
\_def\mathbin{Bin}
\_def\mathrel{Rel}
\_def\mathopen{Open}
\_def\mathclose{Close}
\_def\mathpunct{Punct}
\_def\mathaccent{Acc}
\_def\mathaccentwide{Accw}
\_def\mathbotaccentwidth{AccBw}
\_def\mathbotaccent{AccB}
\_def\mathaccentoverlay{AccO}
\_def\mathover{Over}
\_def\mathunder{Under}
\_typosize[\[7.5/9\]]\_normalmath \_everymath={}
\_medskip\_goodbreak
\_tobegin
\_text{Codes U+00000 \_dots\ U+10000}
\_text{\begmulti 3}
\_text{\input unimath-table.opm}
\_text{\endmulti}
\_text{\endgroup}
\_medskip\_goodbreak
\_tobegin
\_text{Codes U+10001 \_dots\ U+1EEF1 \_\let\UnicodeMathSymbol=\UnicodeMathSymbolA}
\_text{\begmulti 4}
\_text{\input unimath-table.opm}
\_text{\endmulti}
\_text{\endgroup}
\end{verbatim}

2.17 Scaling fonts in document (high-level macros)

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

\begin{verbatim}
\_codedecl \_tobegin \_text{\_typosize [⟨font-size⟩]/⟨baselineskip⟩] sets given parameters. It sets text font size by the \_setfontsize macro and math font sizes by setting internal macros \_sizemtext, \_sizemscript and \_sizemsscript. It uses common concept font sizes: 100\%, 70\% and 50\%. The \_setmainvalues sets the parameters as main values when the \_typosize is called first.}
\_tobegin
\_tobegin
\_protected\_def \_typosize \_begin{verbatim}
\_setfontsize\_mathfontsize \_setbaselineskip \_setmainvalues \_ignorespaces
\end{verbatim}
\_protected\_def \_textfontsize\_if\_else \_setfontsize\_ptunit\_fi
\end{verbatim}
\end{verbatim}
\def \mathfontsize #1{\if$#1$\else \tmpdim=#1\ptunit \edef\sizemtext{\ea\ignorept \the\tmpdim \ptmunit}\fi \t}\edef\sizemscript{\ea\ignorept \the\tmpdim \ptmunit}\t \edef\sizemsscript{\ea\ignorept \the\tmpdim \ptmunit}\t \fi}

\public \typosize ;

\typosize [⟨font-factor]/⟨baseline-factor⟩] scales font size and baselineskip by given factors in respect to current values. It calculates the \typosize parameters and runs the \typosize.

\protected\def \typoscale [#1/#2]{\ifx$#1$\def\tmp{[/}\else \settmpdim{#1}\optsize \edef\tmp{[/\ea\ignorept \the\tmpdim/}\fi \ifx$#2$\edef\tmp{\tmp]\}\else \settmpdim{#2}\baselineskip \edef\tmp{\tmp \ea\ignorept \the\tmpdim]\}\fi \ea \typosize \tmp}

\public \typoscale ;

\def \setbaselineskip #1{\if$#1$\else \tmpdim=#1\ptunit \baselineskip=\tmpdim \relax \bigskipamount=\tmpdim plus.33333\tmpdim minus.33333\tmpdim \medskipamount=.5\tmpdim plus.16666\tmpdim minus.16666\tmpdim \smallskipamount=.25\tmpdim plus.08333\tmpdim minus.08333\tmpdim \normalbaselineskip=\tmpdim \jot=.25\tmpdim \maxdepth=.33333\tmpdim \setbox\strutbox=\hbox{\vrule height.709\tmpdim depth.291\tmpdim width0pt}\fi}

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

- 0: no math fonts are loaded. User must use \normalmath or \boldmath explicitly.
- 1: \normalmath is run if \typosize/\typoscale are used first or they are run at outer group level. No \everymath/\everydisplay are set in this case. If \typosize/\typoscale are run repeatedly in a group then \normalmath is run only when math formula occurs. This is done using \everymath/\everydisplay and \_setmathfonts. \mfontsrule=1 is default.
- 2: \normalmath is run whenever \typosize/\typoscale are used. \everymath/\everydisplay registers are untouched.

\def \setmainvalues {%

\newskip \mainbaselineskip \mainbaselineskip=0pt \relax
\newdimen \mainfosize \mainfosize=0pt
\newcount \mfontsrule \mfontsrule=1
\def \setmainvalues {%
\mainbaselineskip=\baselineskip

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

If \textfontrule=1 then \textsize does not load math fonts immediately but at the first math formula. It is done by \everymath register, but the contents of this register is processed inside the math group. If we do \everymath=\textnormalmath then this complicated macro will be processed three times in your example above. We want only one processing, so we do \everymath=\textsetmathfonts and this macro closes math mode first, loads fonts and opens math mode again.

\textsize and \textfontscale do modification of the size of the current font. They are implemented by the \newcurrfontsizes macro.

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

The \boldify macro does \let\it\bf, \let\bf\it and \let\normalmath=\boldmath. All following text will be in bold. If should be used after \textsize or \textfontscale macros.

The internal \runboldmath macro runs \boldmath immediately if no delay of the math font loading

102
is set by \_setmainvaluesL.
The \rm, \it in math mode must keep its original meaning.

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

2.18 Output routine

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

- \begoutput which does:
  - increments \pageno,
  - prints \Xpage{{\pageno}} to the .ref file (if \openref is active),
  - calculates \hoffset,
  - sets local meaning of macros used in headlines/footlines (see \regmacro).

- \shipout \completepage, which is \vbox of –
  - background box, if \pgbackground is non-empty,
  - headline box by \makeheadline, if the \headline is nonempty,
  - \vbox to\vsize of \pagecontents which consists of –
    - \pagedest, the page destination pg:{\pageno} for hyperlinks is created here,
    - \topins box if non-empty (from \topinserts),
    - \box255 with completed vertical material from main vertical mode,
    - \footnoterule and \footins box if nonempty (from \fnote, \footnote),
    - \pgbottomskip (default is 0pt).
  - footline box by \makefootline, if the \footline is nonempty

- \endoutput which does:
  - increments \pageno using \advancepageno
  - runs output routine repeatedly if \dosupereject is activated.

\optexoutput is the default output routine. You can create another...

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

The \prepoffsets can set \hoffset differently for the left or right page. It is re-defined by the \margins macro.
The \_regmark tokens list includes accumulated \#2 from the \_regmacro. Logos and other macros are re-defined here (locally) for their usage in headlines or footlines.

\begin{verbatim}
def \begoutput{\incr\_gpageno
   \_immediate\_wref\_Xpage{\_the\_gpageno}\{\_folio\}\
   \_setxhsize \_prepoffsets \_the\_regmark}
def \_endoutput{\advancepageno
   \_ifnum\outputpenalty>-20000 \_else\dosupereject\fi}
def \_prepoffsets {}
\end{verbatim}

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

\begin{verbatim}
def \_gpageno\_setxhsize {\_global\_xhsize=\_hsize \_global\_let\_setxhsize=\_relax}
def \_makeheadline {\_istoksempty \_headline \iffalse
   \vbox to\zo{\_baselineskip=\headlinedist \lineskiplimit=-\maxdimen
      \_hbox to\xhsize{\_the\_headline}\_hbox()}\_nointerlineskip
   \fi}
def \_makefootline {\_istoksempty \_footline \iffalse
   \_baselineskip=\footlinedist \_lineskiplimit=-\maxdimen
   \_hbox to\xhsize{\_the\_footline}\_hbox()\_nointerlineskip
   \fi}
\end{verbatim}

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

The \_ensureblack sets the typesetting of its parameter locally to \Black color. We needn’t do this if colors are never used in the document. So, the default value of the \_ensureblack macro is empty. But the first usage of color macros in the document re-defines \_ensureblack. See the section2.20 for more details.

\begin{verbatim}
def \_completepage{\vbox{\_istoksempty \_pgbackground
   \_iffalse \_ensureblack{\_backgroundbox{\_the\_pgbackground}}\_nointerlineskip \fi
   \_vbox to\zo{\_baselineskip=\headlinedist \lineskiplimit=-\maxdimen
      \_hbox to\xhsize{\_the\_headline}\_hbox{}}\_nointerlineskip
   \_vbox to\zo{\_baselineskip=\footlinedist \lineskiplimit=-\maxdimen
      \_hbox to\xhsize{\_the\_footline}}\_nointerlineskip
   \_vbox{\_pagecontents}\_nointerlineskip
   \_makeheadline\_makefootline}
\end{verbatim}

The \_pagecontents is similar as in plain \TeX. The only difference is that the \_pagedest is inserted at the top of \_pagecontents and \_ensureblack is applied to the \_topins and \_footins material. The \_footnoterule is defined here.
The `\topins` macros `\topinsert`, `\midinsert`, `\pageinsert`, `\endinsert` are the same as in plain \TeX. The `\footnote` and `\footstrut` are defined as in plain \TeX.

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.

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 0 pt by default.

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.

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

2.19 Margins

The \margins macro is documented in the section 1.2.1.

\margins/pg \margins takes its parameters, does calculation and sets \hoffset, \voffset, \hsize and \vsize registers. Note that OpTEX sets the page origin at the top left corner of the paper, no at the obscure position 1in, 1in. It is much more comfortable for macro writers.
The common page dimensions are defined here.

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

2.20 Colors

The colors have different behavior than fonts. Marks (whatsits) with color information are stored into PDF output and \TeX\ doesn’t interpret them. The PDF viewer (or PDF interpreter in a printer) reads these marks and switches colors according to them. This is independent of \TeX\ group mechanism. You can declare \\nolocalcolor at the beginning of the document, if you want this behavior. In this case, if you set a color then you must return to the black color using \Black manually.

By default, Op\TeX\ sets \localcolor. It means that the typesetting returns to a previous color at the end of the current group, so you cannot write \Black explicitly. This is implemented using the \aftergroup feature. There is a limitation of this feature: when a color selector is used in a group of a box, which is saved by \setbox, then the activity or reconstruction of the previous color is processed at \setbox time, no in the box itself. You must correct it by double group:

\setbox0=\hbox{\Red text} % bad: \Black is done after \setbox \setbox0=\hbox{\{\Red text\}} % good: \Black is done after group inside the box

The implementation of colors is based on colorstack, so the current color can follow across more pages. It is not so obvious because PDF viewer (or PDF interpreter) manipulates with colors locally at each PDF page and it initializes each PDF page with black on white color.

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 $\setgreycolor\{\langle Grey \rangle\}$ should be used in color selectors or user can specify these macros explicitly.

The color mixing processed by the $\colordef$ is done in the subtractive color model CMYK. If the result has a component greater than 1 then all components are multiplied by a coefficient in order to the maximal component is equal to 1.

You can move a shared amount of CMY components (i.e. their minimum) to the $K$ component. This saves the color toners and the result is more true. This should be done by $\useK$ command at the end of a linear combination used in $\colordef$. For example

$\colordef \myColor{.3\Green + .4\Blue \useK}$
The \usek command exactly does:

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

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

\colordef \Color {Black\-Brown}

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{Green+0.6\-Yellow} % the same as \colordef\mycolor{Green+0.6\-Yellow}

The \rgbcolordef can be used to mix colors in additive color model RGB. If \onlyrgb is declared, then \colordef works as \rgbcolordef.

If a CMYK to RGB or RGB to CMYK conversion is needed then the following simple formulae are used (ICC profiles are not supported):

CMYK to RGB:
\[ R = (1 - C)(1 - K), \quad G = (1 - M)(1 - K), \quad B = (1 - Y)(1 - K). \]

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

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

We declare internal boolean value \_iflocalcolor ad do \localcolor as default.

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

By default, the \setcmykcolor \setrgbcolor and \setgreycolor macros with {⟨components⟩} parameter expand to \setcolor{⟨pdf-primitive⟩} using \_formatcmyk or \_formatrgb or \_formatgrey expandable macros. For example \setcmykcolor{1 0 0} expands to \setcolor{1 0 0 rg 1 0 0 RG}. We set both types of colors (for lines (K or RG or G) and for fills (r or rg or g) together in the ⟨pdf-primitive⟩ command. This is the reason why the \fillstroke uses both its parameters. If only fills are needed you can do \def\fillstroke{1#2[#1]}. If only strokes are needed you can do \def\fillstroke{1#2[#2]}. 

108
The \_onlyrgb declaration redefines \_formatcmyk in order it expands to its conversion to RGB \textit{⟨pdf-primitive⟩}. This conversion is done by the \_cmyktorgb macro. Moreover, \_onlyrgb re-defines three basic RGB colors for RGB color space and re-declares \colordef as \rgbcolordef. The \onlycmyk macro does similar work, it re-defines \_formatrgb macro. The Grey color space is unchanged and works in both main settings (RGB or CMYK) without collisions.

The \_setcolor macro redefines empty \_ensureblack macro (used in output routine for headers and footers) to \_ensureblackA which sets Black at the start of its parameter and returns to the current color at the end of its parameter.

The current color is saved into \_currentcolor macro and colorstack is pushed. Finally, the \_colorstackpop is initialized by \aftergroup if \localcolor is declared.

You can save the current color to your macro by \let \yourmacro = \_currentcolor and you can return to this color by the command \_setcolor \yourmacro.

The colorstack is initialized here and the basic macros \_colorstackpush, \_colorstackpop and \_colorstackset are defined here.

We need to open a special color stack for footnotes because footnotes can follow on the next pages and their colors are independent of colors used in the main page-body. The \_openfnotestack is defined as \_openfnotestackA when the \_setcolor is used first. The \_fnotestack is initialized in in \everyjob because the initialization is not saved to the format.

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 \textit{[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.
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 \(\_\text{stripzeros}\) removes these trailing zeros at the expand processor level. So \(\text{\_stripzeros} 0.300 0.400 0.560\); expands to \(0.3 .4 .56\).

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

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

\texttt{\_addcolor} \(\langle\text{coef}\rangle!\langle\text{mod}\rangle\langle\text{type}\rangle\) expands to \texttt{\_addcolor:}\(\langle\text{mod}\rangle\langle\text{type}\rangle\langle\text{coef}\rangle\) for example it expands to \texttt{\_addcolor:}\(\langle R\rangle\langle K\rangle\langle\text{coef}\rangle\) followed by one or three or four numbers (depending on \(\langle\text{type}\rangle\)). \(\langle\text{mod}\rangle\) is = (use as is) or ~ (use complementary color). \(\langle\text{type}\rangle\) is \(K\) for CMYK, \(R\) for RGB and \(G\) for GREY color space. Uppercase \(\langle\text{type}\rangle\) informs that \texttt{\_cmykcolordef} is processed and lowercase \(\langle\text{type}\rangle\) informs that \texttt{\_rgbcolordef} is processed. All variants of commands \texttt{\_addcolor:}\(\langle\text{mod}\rangle\langle\text{type}\rangle\) are defined. All of them expand to \texttt{\_addcolorA} \(\langle v1\rangle \langle v2\rangle \langle v3\rangle \langle v4\rangle\) which adds the values of Lua variables. The \texttt{\_rgbcolordef} uses \texttt{\_addcolorA} \(\langle R\rangle \langle G\rangle \langle B\rangle \langle 0\rangle\) and \texttt{\_cmykcolordef} uses \texttt{\_addcolorA} \(\langle C\rangle \langle M\rangle \langle Y\rangle \langle K\rangle\). So the Lua variable names are a little confusing when \texttt{\_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 \(\langle\text{what-define}\rangle\) is defined as \(\langle\text{selector}\rangle\{\langle\text{expanded }\_\text{tmpb}\rangle\}\), for example \_setcmykcolor{1 0 .5 .3}.

\begin{verbatim}
\def\rgbcolordef{(\_commoncolordef \setrgbcolor krg)}
\def\cmykcolordef{(\_commoncolordef \setcmykcolor KRG)}
\def\commoncolordef#1#2#3#4#5#6{%\begingroup \directlua{color_C=0 color_M=0 color_Y=0 color_K=0}% \def\setcmykcolor##1{!=#2 ##1 }% \def\setrgbcolor ##1{!=#3 ##1 }% \def\setgreycolor##1{!=#4 ##1 }% \let\useK=_relax \edef\tmpb{+#6}% \replstring\tmpb{+ }{+}\replstring\tmpb{- }{-}\replstring\tmpb{+}{\_addcolor}\replstring\tmpb{-}{\_addcolor-}\replstring\tmpb{^!=}{!^}\replstring\tmpb{-!}{-1!}\ifx K#2\let\useK=_douseK \fi \_\edef\tmpb{\_colordefFin}% \ifx k#2\edef\tmpb{\_ea\_stripK \_tmpb;}\fi \_ea\_endgroup \_def\ea#5\ea{(\_ea#1\ea{\_tmpb})}% \_def\addcolor#1!#2#3{\cs{addcolor:#2#3}#1}
\_def\addcolorA #1 #2 #3 #4 #5{\def\tmpa{#1}\ifx\tmpa\empty\else\edef\tmpa{\_tmpa*}\fi \directlua{color_C=math.max(color_C+\_tmpa#2,0)
color_M=math.max(color_M+\_tmpa#3,0)
color_Y=math.max(color_Y+\_tmpa#4,0)
color_K=math.max(color_K+\_tmpa#5,0)}}\_sdef{addcolor:=K}#1 #2 #3 #4 #5{\_addcolorA #1 (1-#2) (1-#3) (1-#4) #5}\_sdef{addcolor:^K}#1 #2 #3 #4 #5{\_addcolorA #1 0 0 0 (1-#5)}\_sdef{addcolor:=G}#1 #2 #3 #4 #5 {\_addcolorA #1 0 0 (1-#4) #5}\_sdef{addcolor:=r}#1 #2 #3 #4 #5 {\_addcolorA #1 #2 #3 #4 0}\_sdef{addcolor:^r}#1 #2 #3 #4 #5 {\_addcolorA #1 (1-#2) (1-#3) (1-#4) 0}\_sdef{addcolor:=k}#1 #2 #3 #4 #5 #6\_edef\tmpa{\_noexpand\_addcolorA #1 \_cmyktorgb #2 #3 #4 #5 #6}% \_sdef{addcolor:^k}#1 #2 #3 #4 #5 #6\_cs{addcolor:=k}#1 (1-#2) (1-#3) (1-#4) (1-#5) (1-#6)}\_let\_colordef=\_cmykcolordef % default \_colordef is \_cmykcolordef
\end{verbatim}

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

\begin{verbatim}
\def\useK\{\_useK\}
\def\colordef\{\_colordef\}
\public \_cmykcolordef \_rgbcolordef;
\end{verbatim}

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

\begin{verbatim}
def\vr{\vrule height10pt depth2pt width20pt}
def\showcolor{\hbox{\tt \_tmp:\ \_csname \_tmp\endcsname \vr}\space\space}
begmulti 4 \typosize[11/14] \_morecolors \endmulti
\end{verbatim}

111
2.21 The .ref file

The .ref file has the name \jobname.ref and it saves information about references, TOC lines, etc. All data needed in next \TeX run are saved here. \OpTeX reads this file at the beginning of the document (using \everyjob) if such file exists. The .ref file looks like:

\Xrefversion{(ref-version)}
\_Xpage{(gpageno)}{(pageno)}
\_Xtoc{(level)}{(type)}{(title)}
\_Xlabel{(label)}{(text)}
\_Xlabel{(label)}{(text)}
\_Xlabel{(label)}{(text)}

where (gpageno) is internal page number globally numbered from one and (pageno) is a page number (\the\pageno) used in pagination (they may differ). Each page begins with \_Xpage. The (label) is a label used by user in \label\[⟨label⟩] and (text) is a text which should be referenced (the number of section or table, for example 2.3.14). The (title) is the title of the chapter ((level)=1, (type)=chap), section ((level)=2, (type)=sec), subsection ((level)=3, (type)=secc). The \_Xpage is written at the beginning of each page, the \_Xtoc is written when chapter or section or subsection title exists on the page and \_Xlabel when labeled object prefixed by \label\[⟨label⟩] exists on the page.

The .ref file is read when the processing of the document starts using \everyjob. It is read, removed, and opened to writing immediately. But the .ref file should be missing. If none forward references are needed in the document then .ref file is not created. For example, you only want to test a simple plain \TeX macro, you create test.tex file, you do optex test and you don’t need to see an empty test.ref file in your directory.

The \_inputref macro is used in \everyjob. It reads \jobname.ref file if it exists. After the file is read then it is removed and opened to write a new contents to this file.

If the file does not exist then it is not created by default. It means that if you process a document without any forward references then no \jobname.ref file is created because it is unusable. The \_wref macro is a dummy in this case.
If a macro needs to create and to use .ref file then such macro must use \openref. When the file is created (using internal \_openref) then the \_wref{\{macro\}{\{data\}}} is redefined in order to save the line \{\{macro\}{\{data\}}} to the .ref file using asynchronous \write primitive. Finally, the \_openref destroys itself, because we need not open the file again.

The \_wref{\{csname\}{\{params\}}} does exactly \write\_reffile{\{string\}{\{csname\}{\{params\}}} in this case and \_ewref{\{csname\}{\{params\}}} does \write\_reffile{\{string\}{\{expanded-params\}}}.

We are using the convention that the macros used in .ref file are named \_X\{foo\}. If there is a new version of OpTeX with a different collection of such macros then we don’t want to read the .ref files produced by an old version of OpTeX or by OPmac. So the first line of .ref file is in the form

\Xrefversion{\{version\}}

We can check the version compatibility by this macro. Because OPmac does not understand \_Xrefversion we use \Xrefversion (with a different number of \{version\}) form OPmac here. The result: OPmac skips the .ref files produced by OpTeX and vice versa.

You cannot define your special .ref macros before .ref file is read because it is read in \everyjob. But you can define such macros using \refdecl{\{definitions of your ref macros\}}. This command sends to .ref file your \{definitions of your ref macros\} immediately. Next lines in .ref file should include our macros. Example from CTUstyle2:

\refdecl{
  \def\totlist{} \def\toflist{}^^J
  \def\Xtab\#1\#2\#3{\addto\totlist{\totline{\#1}{\#2}{\#3}}}^^J
  \def\Xfig\#1\#2\#3{\addto\toflist{\tofline{\#1}{\#2}{\#3}}}}

We must read \{definition of your ref macros\} when the catcode of \# is 12 because we needn’t duplicate each \# in the .ref file.

2.22 References

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

\Xpage{\{gpageno\}}{\{gpageno\}} saves the parameter pair into \currpage. Resets \lfnotenum; it is used if footnotes are numbered from one at each page.
Counter for the number of unresolved references \_unresolvedrefs.

\_newcount\_unresolvedrefs \_unresolvedrefs=0

\_Xlabel \{⟨label⟩\}⟨text⟩ saves the \_lab:\langlelabel⟩ and saves \_pgref:\{⟨label⟩\} to \_pgref:\{⟨label⟩\}.

\_def\_Xlabel#1#2{\_sdef\_lab:#1{#2}\_sxdef\_pgref:#1{\_ea\_bracketspg\_curpage}}

\_def\_bracketspg#1#2{[pg:#1]{#2}}

\label\{⟨label⟩\} saves the declared label to \_lastlabel and \wlabel\{⟨text⟩\} uses the \_lastlabel and activates \_wref\ \_Xlabel\{⟨label⟩\}⟨text⟩.

\def\_label[#1]{\_isempty{#1}\_iftrue \_global\_let \_lastlabel=\_undefined
\_else \_isdefined{l0:#1}\_iftrue \_slideshook\_opwarning{Duplicated label \#1, ignored}\_else \_xdef\_lastlabel{#1}\_fi
\_fi\_ignorespaces
\_let \_slideshook=\_relax % redefined if \slides + \slideshow.
\def\_wlabel#1{\_ifx\_lastlabel\_undefined \_else
\_dest[ref:\_lastlabel]\%\_printlabel\_lastlabel
\_ewref \_Xlabel\{⟨label⟩\}⟨text⟩
\_sxdef\_lab:\_lastlabel{#1}\_sxdef\l0:\_lastlabel{\_undefined}
\_global\_let\_lastlabel=\_undefined\_fi
\_fi
\_fi
\_public \label \_wlabel ;

\ref\{⟨label⟩\} uses saved \_lab:\langlelabel⟩ and prints (linked) ⟨text⟩. If the reference is backward then we know \_lab:\langlelabel⟩ without any need to read REF file. On the other hand, if the reference is forwarded, then we don’t know \_lab:\langlelabel⟩ in the first run of \TeX and we print a warning and do \_openref.

\pgref\{⟨label⟩\} uses \_sxdef\_{\_lab}\{⟨label⟩\}\_{\_pageno} from \_pgref\{⟨label⟩\} and prints (linked) \{ ⟨pageno⟩ \} using \_ilink macro.

\def\_ref[#1]{\_isdefined{\_lab:#1}\_iftrue \_global\_let \_lastlabel=\_undefined
\_else \_isdefined{10:#1}\_iftrue \_slideshook\_opwarning{Duplicated label \#1, ignored}\_else \_xdef\_lastlabel{#1}\_fi
\_fi\_ignorespaces
\_let \_slideshook=\_relax % redefined if \slides + \slideshow.
\def\_pgref[#1]{\_isdefined{\_pgref:#1}\_iftrue \_ea\_ea\_ea\_ilink \_csname \_pgref:#1\_endcsname
\_else ??\_opwarning{pg-label \#1 unknown. Try to TeX me again}\_fi
\_fi
\_fi
\_public \ref \_pgref ;

Default \_printlabel is empty macro (labels are not printed). The \_shoulalbes redefines it as box with zero dimensions and with left lapped \{ ⟨label⟩ \} in blue 10pt \tt font shifted up by 1.7ex.
2.23 Hyperlinks

There are four types of internal links and one type of external link:

- \ref{label} – the destination is created when \label{[label]} is used, see also the section 2.22.
- \toc{tocrefnum} – the destination is created at chap/sec/secc titles, see also the section 2.24.
- \pg{pgapeno} – the destination is created at beginning of each page, see also the section 2.18.
- \cite{bibpart}/(bibnum) – the destination is created in bibliography reference, see section 2.32.1.
- \url{[url]} – used by \url or \ulink, see also the end of this section.

The \tocrefnum, \pgapeno, and \bibnum are numbers starting from one and globally incremented by one in the whole document. The registers \tocrefnum, \pgapeno and \bibnum are used for these numbers.

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

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

\def\dest[[type]:[spec]]\{\text\} creates an internal link to \dest with the same \{type\}:\{spec\} but only one \dest in the document. If \hyperlinks command is not used, then \link only prints \{text\} else its meaning is set to \linkactive. The \linkactive is implemented by \pdfstartlink..., \pdfendlink primitives. The \color is the color of the link text generated by the \linkactive macro. The \color parameter can be overwritten by definition of \\{type\}linkcolor. For example \def\toclinkcolor{\Red} means that links from table of contents are in red. This is similar concept as \def\tocborder.

\link[type]{\{spec\}}\{\text\} is equivalent to \link but the \color is used from \hyperlinks declaration (or it is overwritten by \def\\{type\}linkcolor).

\linkdimens are default dimensions of the link area.

\url{[url]}\{\text\} creates external link. It prints only the \{text\} by default but the \hyperlinks declaration defines it as \ulink{[url]}\{\text\}. The external link is created by the \pdfstartlink..., \pdfendlink primitives. The \url is detokenized with \escapechar=-1 before it is used, so \verb\url\# etc. can be used in the \url.

\def\pdfstartlink \linkdimens attr{\pdfborder{[#1]}}
\def\pdfendlink{\pdfborder{[#1]}}

The \_pdfstartlink primitive uses attr{\_pdfborder{/type}} in its parameter (see \_linkactive or \_urlactive macros). The \_pdfborder{/type} macro expands to /C[? ? ?] /Border[0 0 .6] if the \_pdfborder macro (i.e. \_refborder, \_citeborder, \_tocborder, \_pgborder, \_urlborder, \_fntborder or \_fnfborder) is defined. Users can define it in order to create colored frames around active links. For example \def\_tocborder{1 0 0} causes red frames in TOC (not printed, only visible in PDF viewers).

\hyperlinks{⟨ilink_color⟩}{⟨ulink_color⟩} activates \dest, \link, \ilink, \ulink in order they create links. These macros are redefined here to their “active” version.

\url{⟨url⟩} does approximately the same as \ulink{⟨url⟩}{⟨url⟩}, but more work is done before the \ulink is processed. The link-version of ⟨url⟩ is saved to \_tmpa and the printed version in \_tmpb. The printed version is modified in order to set breakpoints to special places of the ⟨url⟩. For example // is replaced by \_urlskip/\_urlskip/\_urlbskip where \_urlskip adds a small nonbreakable glue between these two slashes and before them and \_urlbskip adds a breakable glue after them.

The text version of the ⟨url⟩ is printed in \_urlfont.

2.24 Making table of contents

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

- ⟨level⟩: 0 reserved, 1: chapter, 2: section, 3: subsection
- ⟨type⟩: the type of the level, i.e. chap, sec, secc
- ⟨number⟩: the number of the chapter/section/subsection in the format 1.2.3
- ⟨o-title⟩: outlines title, if differs from ⟨title⟩.
- ⟨title⟩: the title text
- ⟨gpageno⟩: the page number numbered from 1 independently of pagination
- ⟨pageno⟩: the page number used in the pagination
The last two parameters are restored from previous \_Xpage\{pageno\}\{gpageno\}, data were saved in the \_currpage macro.

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

The auxiliary macros are:

- \_llaptoclink\langle text \rangle does \_noindent \_llap\langle linked text \rangle.
- \_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 records whith rapped \langle pageno \rangle.
- \_pgn\langle pageno \rangle creates \langle pageno \rangle as link to real \langle gpage \rangle saved in \#6 of \_tocline. This is temporarily defined in the \_tocline.

If you want a special formatting of TOC with adding more special lines (no generated as titles from \chap, \sec, \secc), you can define \addtotoc\langle level \rangle\{\langle type \rangle\}\{\langle number \rangle\}\{\langle a-title \rangle\}\{\langle title \rangle\} macro:
and you can declare special lines (or something else) as an unused level (10 in the following example):
\addtotoc{10}{blue-line}{}{\relax}{⟨ blue text to be added in the TOC ⟩}

Now, users can add a blue line into TOC by anywhere in the document. Note that \relax in the fourth parameter means that outline will be not generated. And second parameter blue-line is only a comment (unused in macros).

\maketoc prints warning if TOC data is empty, else it creates TOC by running \_toclist

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

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:\_count0:\_count1:\count2 etc. For example, when section (level 2) is processed in the first step then we do:

\advance \count2 by 1
% increases the mapping pointer of the type
\advance \_ol:\_count0:\_count1:\count2 by 1
% increases the number of descendants connected
% to the parent of this section.

When the second step is processed, then we only read the stored data about the number of descendants. And we use it in count parameter of \_pdfoutline primitive.
For linking, we use the same links as in TOC, i.e. the `toc:\_the\_tocrefnum` labels are used. `\insertoutline {⟨text⟩}` inserts one outline with zero direct descendants. It creates a link destination of the type `oul:\num` into the document (where `\insertoutline` is used) and the link itself is created too in the outline.

```latex
\texttt{\_codedecl \outlines \{PDF outlines <2021-02-09}\} \% preloaded in format
\begin{verbatim}
\_def\_outlines\#1{\_pdfdoccatalog{\_PageMode}{UseOutlines}\_openref
\_if\_toclist\_empty
\_opwarning{\_noexpand\outlines -- data unavailable. TeX me again}\
\_incr\_unresolvedrefs
\_else
\_if\_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
}\_fi}
\_def\_outlinesA#1#2#3#4#5#6#7{\_ifcsname\_outlinelevel\relax
\_isequal{\relax}{#4}\_iffalse
\_advance\_count#1 by1
\_ifcase#1\_or
\_addoneol{oul:\_the\_count0}\_or
\_addoneol{oul:\_the\_count0\:\_the\_count1}\_or
\_addoneol{oul:\_the\_count0\:\_the\_count1\:\_the\_count2}\_or
\_addoneol{oul:\_the\_count0\:\_the\_count1\:\_the\_count2\:\_the\_count3}\_fi
\_fi}
\_def\_addoneol#1{\_ifcsname\_outlinelevel\_\relax
\_tmpnum=\_csname\_\_\_\relax
\_advance\_tmpnum by1
\_sxdef{\_outlinelevel}{\_the\_tmpnum}\_else\_sxdef{\_outlinelevel}{1}_\fi}
\_def\_outlinesB#1#2#3#4#5#6#7{\_advance\_tocrefnum by1
\_isequal{\relax}{#4}\_iffalse
\_advance\_count#1 by1
\_ifcase#1\_or
\_tmpnum=\_trycs{oul:\_the\_count0}{0}\_or
\_tmpnum=\_trycs{oul:\_the\_count0\:\_the\_count1}{0}\_or
\_tmpnum=\_trycs{oul:\_the\_count0\:\_the\_count1\:\_the\_count2}{0}\_or
\_tmpnum=\_trycs{oul:\_the\_count0\:\_the\_count1\:\_the\_count2\:\_the\_count3}{0}\_or
\_tmpnum = 0 \_relax\_fi
\_isempty{#4}\_iftrue \_pdfunidef\_tmp{#5}\_else \_pdfunidef\_tmp{#4}\_fi
\_outlinesC{toc:\_the\_tocrefnum}{\_ifnum#1<\_outlinelevel\_space\else-\fi}{\_tmpnum}{\_tmp}\_fi}
\_def\_outlinesC#1#2#3#4\{\_pdfoutline goto name{#1} count #2#3\{#4\}\_relax}
\_def\_addoneol#1{\_ifcsname\_outlinelevel\_\relax
\_tmpnum=\_csname\_\_\_\relax
\_advance\_tmpnum by1
\_sxdef{\_outlinelevel}{\_the\_tmpnum}\_else\_sxdef{\_outlinelevel}{1}_\fi}
\_def\_outlines#1\{\_pdfdest name{oul:\_the\_count0} xyz\_relax
\_pdfunidef\_tmp{#1}\_pdfoutline goto name{oul:\_the\_count0} count0 {\_tmp}\_relax
\_fi}
\_public \outlines \insertoutline ;
\end{verbatim}
```

### 2.25.2 Strings in PDF outlines

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

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

\texttt{pdfuni-string.opm}

3 \texttt{\_\_codedecl pdfunidef \{PDFunicode strings for outlines <2021-02-08\} \% preloaded in format}

\texttt{hexprint} is a command defined in Lua, that scans a number and expands to its UTF-16 Big Endian encoded form for use in PDF hexadecimal strings.

\texttt{pdfuni-string.opm}

10 \texttt{\begingroup}
11 \texttt{\_catcode\`\%=12}
12 \texttt{\_gdef hexprint\{\_directlua{ local num = token.scan\_int() if num < 0x10000 then tex.print(string\_format("\%04X", num)) else num = num - 0x10000 local high = bit32\_rshift(num, 10) + 0xD800 local low = bit32\_band(num, 0x3FF) + 0xDC00 tex.print(string\_format("\%04X\%04X", high, low)) end \end\_directlua\}}}
13 \texttt{\egroup}

\texttt{pdfunidef\macro{⟨text⟩}} does more things than only converting to hexadecimal PDF string. The ⟨text⟩ can be scanned in verbatim mode (it is true because \texttt{\_Xtoc} reads the ⟨text⟩ in verbatim mode). First \texttt{\_edef do \_scantextokens\_unexpanded} and second \texttt{\_edef} expands the parameter according to current values on selected macros from \texttt{\_regoul}. Then \texttt{\_removeoutmath} converts ..$x^2$.. to ..$x^2$.., i.e removes dollars. Then \texttt{\_removeoutbraces} converts .{x}.. to .x.. Finally, the ⟨text⟩ is detokenized, spaces are preprocessed using \texttt{\_replstring} and then the \texttt{\_pdfunidefB} is repeated on each character. It calls the \texttt{\_directlua} chunk to print hexadecimal numbers in the macro \texttt{\_hexprint}.

Characters for quotes (and separators for quotes) are activated by first \texttt{\_scatextokens} and they are defined as the same non-active characters. But \texttt{\_regoul} can change this definition.
The \_prepinverb\textit{macro}/\texttt{separator}/\{\textit{text}!text\} e.g. \_prepinverb\tmpb\{aaa |bbb| cccc |dd| ee\} does \texttt{\def\tmpb}{\langle su\rangle \{aaa \space bbb\} \langle cccc \space dd\} \{ ee\}} where \langle su\rangle is \texttt{\scantextokens\unexpanded}. It means that in-line verbatim are not argument of \texttt{\scantextoken}. First \texttt{\def\tmpb} tokenizes again the \langle su\rangle but not the parts which were in the the in-line verbatim.

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

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

- Use \_\texttt{abovetitle}\{\langle penaltyA\rangle\}\{\langle skipA\rangle\} and \_\texttt{belowtitle}\langle \texttt{skipB}\rangle for inserting vertical material above and below the section title. The arguments of these macros are normally used, i.e. \texttt{\abovetitle} inserts \langle penaltyA\rangle\langle skipA\rangle and \_\texttt{belowtitle} inserts \langle skipB\rangle. But there is an exception: if \_\texttt{belowtitle}\langle \texttt{skipB}\rangle is immediately followed by \_\texttt{abovetitle}\langle \texttt{penaltyA}\rangle\langle \texttt{skipA}\rangle (for example section title is immediately followed by subsection title), then only \langle \texttt{skipA}\rangle is generated, i.e. \langle \texttt{skipB}\rangle\langle \texttt{penaltyA}\rangle\langle \texttt{skipA}\rangle is reduced only to \langle \texttt{skipA}\rangle. The reason for such behavior: we don’t want to duplicate vertical skip and we don’t want to use the negative penalty in such cases. Moreover, \_\texttt{abovetitle}\langle \texttt{penaltyA}\rangle\langle \texttt{skipA}\rangle takes previous whatever vertical skip (other than from \_\texttt{belowtitle}) and generates only greater from this pair of skips. It means that \langle whatever-skip\rangle\langle \texttt{penaltyA}\rangle\langle \texttt{skipA}\rangle is transformed to \langle \texttt{penaltyA}\rangle\texttt{max}(\langle whatever-skip\rangle\langle \texttt{skipA}\rangle).
reason for such behavior: we don’t want to duplicate vertical skips (from \_belowlistsip, for example) above the title.

- Use \_printrefnum[⟨pre⟩⟨ref-num⟩⟨post⟩] in horizontal mode. It prints ⟨pre⟩⟨ref-num⟩⟨post⟩. The ⟨ref-num⟩ is \_thechapnum or \_thesecnum or \_theseccnum depending on what type of title is processed. If \nonum prefix is used then \_printrefnum prints nothing. The macro \_printrefnum does more work: it creates destination of hyperlinks (if \hyperlinks{}{} is used) and saves references from the label (if \label[⟨label⟩] precedes) and saves references for the table of contents (if \maketoc is used).

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

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

The \_sectionlevel is the level of the printed section:

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

The \_chapx initializes counters used in chapters, the \_secx initializes counters in sections and \_seccx initializes counters in subsections. If you have more types of numbered objects in your document then you can declare appropriate counters and do \addto\_chapx{\yourcounter=0 } for example. If you have another concept of numbering objects used in your document, you can re-define these macros. All settings here are global because it is used by \{\globaldefs=1 \_chapx\}. Default concept: Tables, figures, and display maths are numbered from one in each section – subsections don’t reset these counters. Footnotes declared by \fnotenumchapters are numbered in each chapter from one.

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

The `\chap`, `\sec`, and `\secc` macros are implemented here. The `\inchap`, `\insec` and `\insecc` macros do the real work. First, we read the optional parameter `[⟨label⟩]`, if it exists. The `\chap`, `\sec` and `\secc` macro reads its parameter using `\scantoeol`. This causes that they cannot be used inside other macros. Use `\inchap`, `\insec`, and `\insecc` macros directly in such case.

The `\printrefnum` macro is used in `\print*` macros.

Note that the ⟨title-text⟩ is detokenized before `\wref`, so the problem of “fragile macros” from old \LaTeX never occurs. This fourth parameter is not delimited by {…} but by end of line. This gives possibility to have unbalanced braces in inline verbatim in titles.
\thisoutline{⟨text⟩} saves text to the \_theoutline macro. \_printrefnum uses it and removes it.

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

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

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

OptIPX sets \headline={} by default, so no running headings are printed. You can activate the running headings by following code, for example:

\addto\_chapx {\edef\_runningchap {\thechapnum: \_unexpanded\_ea{\_savedtitle}}} \def \_formatead #1\_2{\isempty{#1}|\iffalse #1: #2\fi} \headline = \% \ifodd\pageno \hfil \ea{\_formatead{\_firstmark}{\_unexpanded{#1}}}}
The `\secl{number} \langle title-text \rangle` should be used for various levels of sections (for example, when converting from Markdown to \TeX{}). `\secl{1}` is `\chap`, `\secl{2}` is `\sec`, `\secl{3}` is `\secc` and all more levels (for `{number}>3`) are printed by the common `\_seclp` macro. It declares only a simple design. If there is a requirement to use such more levels then the book designer can define something different here.

```
\def\_seclp#1{%
  \par
  \ifnum\lastpenalty=0 \removelastskip\medskip\fi
  \noindent{\bf #1}\vadjust{\nobreak}\nl\ignorepars
\}
\def\ignorepars{%
  \isnextchar\par{\ignoresecond\ignorepars}{}
}
\public\_secl ;
```

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

```
\_def\caption/#1{%
  \def\tmpa{#1}\\nospaceafter\capA
  \optdef\capA[]{\trylabel\_incaption}
  \def\_incaption{|\bgroup
    \ifcsname \_tmpa\num_endcsname\_ea\_incr\_csname\_tmpa\num\_endcsname\else\_opwarning{Unknown caption /\_tmpa}\fi
    \edef\_thecapnum{\csname \the\tmpa\num\endcsname}\
    \edef\_thecaptitle{\_mtext{\tmpa}}\
    \ea\the\_csname\_everycaption\_tmpa\_endcsname
    \def\par{\_nbpar\egroup}\\let\par=\par
    \cs{\_printcaption\_tmpa}\
  }
  \_def\_cskip{\par\nobreak\medskip} % space between caption and the object
\}
\public\_caption\_cskip ;
```

The `\_printcaptiont` and `\_printcaptionf` macros start in vertical mode. They switch to horizontal mode and use `\_wlabel\_thecapnum` (in order to make reference and hyperlink destination) 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 re-define the `\_captionsep` macro if you want to use another separator.

```
\_def\_printcaptiont%{
  \noindent\_wlabel\_thecapnum{\bf\_thecaptitle~\_thecapnum}\
  \narrowlastlinecentered\_indent\_futurelet\_next\_captionsep
}
\_def\_printcaptionf{%
  \_if\next.\_ea\_bfnext \_else\_if\next:\_ea\_ea\_ea\bfnext
  \_enspace\_fi\_fi
}
```

If you want to declare a new type of `\caption` with independent counter, you can use following lines, where `\_printcaptionf` for Algorithms are declared:

```
\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...
```

125
The default format of \texttt{\caption} text is a paragraph in block narrower by \_iindent and with the last line is centered. This setting is done by the \_narrowlastlinecentered macro.

\begin{verbatim}
def \narrowlastlinecentered#1{\leftskip=#1plus1fil \rightskip=#1plus-1fil \parfillskip=0pt plus2fil\relax}
\end{verbatim}

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

\begin{verbatim}
optdef \eqmark [] {\trylabel \ineqmark}
def \ineqmark {\incr_dnum \ifinner \else \eqno \fi \wlabel \thednum \hbox{\thednum} \par}
\public \eqmark ;
\end{verbatim}

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

\begin{verbatim}
newcount \counterA \newcount \counterB \newcount \counterC \newcount \counterD \newcount \counterE
\resetABCDE \counterA=0 \counterB=0 \counterC=0 \counterD=0 \counterE=0
\def \theAnum {\chapnum . \secnum . \counterA} \def \theBnum {\chapnum . \secnum . \counterB} \def \theCnum {\chapnum . \secnum . \counterC} \def \theDnum {\chapnum . \secnum . \counterD} \def \theEnum {\chapnum . \secnum . \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 \the\tmpb}
\public \numberedpar ;
\end{verbatim}

The \_\texttt{printnumberedpar} \_\texttt{\theXnum} \{\langle name\rangle\} opens numbered paragraph and prints it. The optional parameter is in \_\texttt{\the\opt}. You can re-define it if you need another design.

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

```
def \theorem {\medskip \bgroup \it \numberedpar A{Theorem}} \def \endtheorem {\par \egroup \medskip}
```

```
def \printnumberedpar #1\{\par \noindent \wlabel #1% (\bf #1 \#1 \istokempty\_opt \iffalse \space \_the\_opt \_fi.)\space \ignorespaces}
```

\subsection{2.27 Lists, items}

\_\texttt{aboveliskip} is used above the list of items,
\_\texttt{belowliskip} is used below the list of items and
\_\texttt{interliskip} is used between items.
\_\texttt{listskipA} is used as \_\texttt{listskipamount} at level 1 of items.
\_\texttt{listskipB} is used as \_\texttt{listskipamount} at other levels.
\_\texttt{setlistskip} sets the skip dependent on the current level of items

126
The \itemnum is locally reset to zero in each group declared by \begitems. So nested lists are numbered independently. Users can set initial value of \itemnum to another value after \beitems if they want. Each level of nested lists is indented by the new \iindent from left. The default item mark is \_printitem.

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

The \enditems runs \_isnextchar \par \{ \_noindent \} thus the next paragraph is without indentation if there is no empty line between the list and this paragraph (it is similar behavior as after display math).

\novspaces sets \listskipamount to 0pt.

Various item marks are saved in \item:⟨letter⟩ macros. You can re-define them or define more such macros. The \_style \{letter\} does \_printitem={\item:⟨letter⟩}. More exactly: \begitems does \_printitem=\defaultitem first, then \_style \{letter\} does \_printitem={\item:⟨letter⟩} when it is used and finally, \_startitem alias * uses \_printitem.
\_athe\{(num)\} returns the \(\langle \text{num} \rangle\)'s lowercase letter from the alphabet. 
\_fullrectangle\{(dimen)\} prints full rectangle with given \(\langle \text{dimen} \rangle\).

The \begblock macro selects fonts from footnotes \_fnset and opens new indentation in a group. \endblock closes the group. This is implemented as a counterpart of Markdown’s Blockquotes. Redefine these macros if you want to declare different design. The \texttt{\texttt{OpTeX trick 0031}} shows how to create blocks with grey background splittable to more pages.

\_code\{⟨text⟩\} expands to \texttt{\detokenize\{⟨text⟩\}} when \texttt{\escapechar=-1}. In order to do it more robust when it is used in \texttt{\write} then it expands as noexpanded \texttt{\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 \texttt{-1} and uses \_regoul token list before \_edef.

The \_code is not \_protected because we want it expands to \_unexpanded\{\_code\{\texttt{space}\}\{⟨text⟩\}\} in \_write parameters. This protect the expansions of the \_code parameter (like `\`, `\*` etc.).

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.
\verbchar{char} saves original catcode of previously declared \verbchar{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 \verbtext{text} by \_printinverbatim and closes group. Suppose that \verbchar{"} is used. Then the following work is schematically done:

```
\def \verbchar{\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 \boldef, so users can put a parameter at the same line where \begtt is. This \#1 parameter is used after \everytt parameters settings, so users can change them locally.

The \begtt macro does \_setverb and another preprocessing, sets \endlinechar to \textasciitilde{} 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{}{...}
```

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{} at each end of line and all spaces are active characters (defined as \textasciitilde{}). Other characters have normal category 11 or 12.

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

The \_printverb macro calls \_printverbline{\langle line \rangle} 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 \_printverblinenum is called here.

The \_printverblinenum (\line) expects that it starts in vertical mode and it must do \par to return the vertical mode. The \_printverblinenum is used here: it does nothing when \_ttline<0 else it prints the line number using \_llap.

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

Macro \verbinput uses a file read previously or opens the given file. Then it runs the parameter scanning by \_viscanparameter and \_viscaminus. 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.
\edef\vinolines{\the\tmpnum}
\ifnum\tmpnum<0
\tmpnum=0
\fi
\edef\vidolines{\the\tmpnum}
\doverbinput
\edef\doverbinput{\
\tmpnum=\vinolines
\advance\tmpnum by-\vinolines
\ifnum\tmpnum<0
\openin\vifile={\vifilename}
\global\viline=0
\else
\edef\vinolines{\the\tmpnum}
\fi
\vskip\parskip \ttskip \wipeepar \setxhsize
\begingroup
\ifnum\ttline<-1 \let\printverblinenum=\relax \else \initverblinenum \fi
\setverb \adef{ }{\dsp}\adef\^^I{\t}\parindent=\ttindent \parskip=0pt
\def\t{\hskip \dimexpr\tabspaces em/2\relax}
\protrudechars=0 % disable protrusion
\the\everytt \relax \ttbmp\relax \ttfont
\savemathsb \endlinechar=`^^J \tmpnum=0
\loop \ifnum\ftline<\vinolines \repeat
\edef\ttlinesave{\ttline=\the\ttline}
\ifnum\ttline=-1 \ttline=\viline \fi
\tmpnum=0 \def\tmpb{}\fi
\ifeof\vifile \tmpnum=\vidolines \fi
\loop \ifnum\tmpnum<\vidolines \repeat
\edef\vireadline{\read\vifile to \tmp \incr\viline}
\ea\prepareverbdata \ea\ttbmp\ea{\ttbmp--}]
\catcode`\=10 \catcode`%=9 % used in \commentchars comments
\ea\printverb \ttbmp\end
\global\ttlinesave
\par \restoremathsb
\endgroup
\ttskip
\ismathchar\par{}{\noindent}]
\def\vireadline{\read\vifile to \tmp \incr\viline}
\def\visaveline{\ea\addto\ea\ttbmp\ea{\ttbmp}}
\public \verbinput;
\restoremathsb \restoremathsb pair is used in \texttt{...} or in \verbinput to temporary suppress the \texttt{mathsbon} because we don’t need to print \texttt{\int_a} in verbatim mode if \texttt{\int_a} is really written. The \restoremathsb is defined locally as \texttt{mathsbon} only if it is needed.

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

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

There is one limitation when \TeX interprets the comments declared by \commentchars. Each block of comments is accumulated to one line and then 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 block does not work. The character % is ignored in comments but you can use \% for printing or \% alone for de-activating \_endpar from empty comment lines.

Implementation: The \commentchars(first)(second) redefines the \_testcommentchars used in \_printverb in order to it removes the following \_iftrue and returns \_iftrue or \_iffalse depending on the fact that the comment characters are or aren’t present at the beginning of the tested line. If it is true (\_ifnum expands to ifnum 10>0) then the rest of the line is added to the \_vcomments macro.

The \hicomments is \relax by default but it is redefined by \commentchars in order to keep no-colorized comments if we need to use feature from \commentchars.

The accumulated comments are printed whenever the non-comment line occurs. This is done by \_printcomments macro. You can re-define it, but the main idea must be kept: it is printed in the group, \reloding \_rm initializes normal font, \catcodetable0 returns to normal catcode table used before \verbinput is started, and the text accumulated in \_vcomments must be printed by \_scantextokens primitive.

\_def\_vcomments{}
\_let\_hicomments=\_relax
\_def\_commentchars#1#2{%\_def\_testcommentchars ##1##2##3\_relax ##4\_iftrue{\_ifnum % not closed in this macro
\_ifx #1##1\_ifx#2##21\fi\_fi 0>0
\_ifx\_relax##3\_relax \_addto\_vcomments{\_endgraf}% empty comment=\enfgraf
\_else \_addto\_vcomments{##3 }\_fi}%
\_def\_hicomments{\_replfromto{\b
#1#2}{^^J}{\w{#1#2####1}^^J}}% used in \hisyntax
\_def\_testcommentchars #1\_iftrue{\_iffalse} % default value of \_testcommentchars
\_def\_printcomments{\_ttskip
\_catcodetable0 \_reloding \_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 \hisytnax 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.

\_def \_visibleisp{\_ifx\_initunifonts\_relax \_def\_dsp{\_char9251 }%}
\_else \_def\_dsp{\_char32 }\_fi
\_let\_dsp=\ % primitive "direct space"
\_public \visibleisp ;

2.28.2 Listings with syntax highlighting
The user can write

\begtt \hisytnax{C}
...
\endtt

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

```latex
\_codedecl \_hisyntaxc {Syntax highlighting for C sources <2020-04-03>}
\_newtoks \_hisyntaxc \_newtoks \_hicolorsc
\_global\_hicolorsc={% colors for C language
\_hicolor K \Red % Keywords
\_hicolor S \Magenta % Strings
\_hicolor C \Green % Comments
\_hicolor N \Cyan % Numbers
\_hicolor P \Blue % Preprocessor
\_hicolor O \Blue % Non-letters
}
\_global\_hisyntaxc={%
\_the\_hicolorsc
\_let\c=\_relax \_let\e=\_relax \_let\o=\_relax
\_replfromto {\*/}{{*/}} {\x C{\#/1}}% /*...*/
\_replfromto {\('\)\{-\}}{{\('\)\{-\}} {\x C{\#/1}}% //...}
\_replthis {\_string\"} {\_string\"} {\z P{\##1}}% #include ...
\_replfromto {\_string\"} {\_string\"} {\z P{\##1}}% " protected inside strings
\_replfromto {\_string\"} {\_string\"} {\z P{\##1}}% "..."}
\_edef\_tmpa {()
\_string{+-*/=\["]\<,>:;\_pcent\_string&\_string^|!?}% non-letters
\_ea \_foreach \_tmpa \_do {\_replthis{#1}{\n\o#1\n}}
\_foreach % keywords
{auto}{break}{case}{char}{continue}{default}{do}{double} %
{else}{enum}{extern}{float}{for}{goto}{if}{int}{long}{register}%
{return}{short}{sizeof}{static}{struct}{switch}{typedef}{union}%
{unsigned}{void}{while}
\_do \_replthis{#1}{\n K{#1}}%
\_replthis{.}{\n .
} % numbers
\_foreach 0123456789
\_do \_replfromto{\n#1}{\n}{\c#1##1\e}
\_replthis{e\e\o\e\o+\c}{e+}
\_replthis{e\e\o\e\o-\c}{e-}
\_edef\o#1{\z O{#1}}
\_edef\c#1\e{\z N{#1}}%
}\}
```

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

User can re-declare colors by `\hicolors={...}` This value has precedence before `\_hicolors⟨name⟩` values declared in the `hicolors-{⟨name⟩}.opm` file. What exactly to do: copy `\_hicolors⟨name⟩={...}` from `hicolors-{⟨name⟩}.opm` to your document, rename it as `\hicolors={...}` and do your own colors modifications.

Another way to set non-default colors is to declare `\newtoks\hicolors⟨name⟩` (without the _ prefix) and set the colors palette here. It has precedence before `\_hicolors⟨name⟩` (with the _ prefix) declared in the `hicolors-{⟨name⟩}.opm` file. This is useful when there are more hi-syntax languages used in one document.

Notes for hi-syntax macro writers

The file `hisyntax-{⟨name⟩}.opm` is read only once in the \TeX\ group. If there are definitions then they must be declared as global.

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

The verbatim text is prepared by pre-processing phase, then the `\_hisyntax⟨name⟩` is applied and then 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 \textbackslash n\textbackslash n\textbackslash n, so \textbackslash n(word)\textbackslash n should be a pattern to finding whole words (no subwords). The \textbackslash n control sequence is removed in the post-processing phase.
• Each end of line is represented by \textbackslash n--\textbackslash n.
• The \textbackslash start control sequence is added before the verbatim text and the \textbackslash _end control sequence is appended to the end of the verbatim text. These control sequences are removed in the post-processing phase.

Special macros are working only in a group when processing the verbatim text.
• \textbackslash n means noting but it should be used as a boundary of words as mentioned above.
• \textbackslash \textbackslash t means a tabulator. It is prepared as \textbackslash n\textbackslash t\textbackslash n because it can be at the boundary of a word.
• \textbackslash x \textbackslash \langle letter\rangle \textbackslash \{\langle text\rangle\} can be used as replacing text. Suppose the example

```latex
\replfromto{/*}{*/}{\textcolor{red}{\textbackslash \{/*#1*/\}}}
```

This replaces all C comments /*...*/ by \textbackslash x C{(*...*/}. But the C comments may span more lines, i.e. the ~\textbackslash J should be inside it.

The macro \textbackslash x \langle letter\rangle \textbackslash \{\langle text\rangle\} is replaced by one or more \textbackslash z \langle letter\rangle \textbackslash \{\langle text\rangle\} in post-processing phase where each parameter \langle text\rangle of \textbackslash z keeps inside one line. Inside-line parameters are represented by \textbackslash x C{\langle text\rangle} and they are replaced to \textbackslash z C{\langle text\rangle} without any change. But:

```latex
\textbackslash x C{\langle text1\rangle}~\textbackslash J\textbackslash z C{\langle text3\rangle}
```

is replaced by

```latex
\textbackslash z C{\langle text2\rangle}~\textbackslash Jz C{\langle text3\rangle}
```

The \textbackslash z \langle letter\rangle \textbackslash \{\langle text\rangle\} is expanded to \textbackslash z: \langle letter\rangle \textbackslash \{\langle text\rangle\} and if \textbackslash hicolor \langle letter\rangle \textbackslash \langle color\rangle is declared then \textbackslash z: \langle letter\rangle \textbackslash \{\langle text\rangle\} expands to \textbackslash \{\langle color\rangle\langle text\rangle\}. So, required color is activated at all lines (separately) where C comment spans.
• \textbackslash y \textbackslash \{\langle text\rangle\} is replaced by \textbackslash \{\langle text\rangle\} in the post processing phase. It should be used for macros without a parameter. You cannot use unprotected macros as replacement text before the post-processing phase, because the post-processing phase is based on expansion whole verbatim text.
The patterns ⟨from⟩, ⟨to⟩ and ⟨pattern⟩ are not found when they are hidden in braces {...}. Example:
\replfromto{/**/}{*}{\x C{/*#1/*}}
replaces all C comments by \x C{...}. The patterns inside {...} are not used by next usage of \replfromto or \replthis macros.

The \_xscan macro does replacing \x by \z in the post-processing phase. The \x ⟨letter⟩ ⟨text⟩ expands to \_xscan ⟨letter⟩ ⟨text⟩ "~J". If #3 is \_end then it signals that something wrong happens, the (from) was not terminated by legal ⟨to⟩ when \replfromto did work. We must to fix it by the \_xscanR macro.

The \hicolor ⟨letter⟩ ⟨color⟩ defines \_z ⟨letter⟩ ⟨text⟩ as \{⟨color⟩⟨text⟩}. It should be used in the context of \x ⟨letter⟩ ⟨text⟩ macros.

The \hisyntax{⟨name⟩} re-defines default \_prepareverbdata ⟨macro⟩ ⟨verbtext⟩ in order to it does more things: It saves ⟨verbtext⟩ to \_tmpb, appends \n around spaces and "~J characters in pre-processing phase, it opens hisyntax-{⟨name⟩}.opm file if \_hisyntax{⟨name⟩} is not defined. Then \_the\_isyntax{⟨name⟩} is processed. Finally, the post-processing phase is realized by setting appropriate values to \x and \y macros and doing \_edef\_tmpb{\_tmpb}.

Aliases for languages can be declared like this. When \hisyntax{xml} is used then this is the same as \hisyntax{html}.

\sdef{\_hialias:html}{\html}
\sdef{\_hialias:json}{\c}
2.29 Graphics

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

\begin{verbatim}
\newbox\mypic
\setbox\mypic = \hbox{\picw=3cm \inspic\{\textit{picture}\}}
\end{verbatim}

My picture: \texttt{\copy\mypic}, again my picture: \texttt{\copy\mypic}, etc.

This code downloads the picture data to the PDF output only once (when \texttt{\setbox} is processed). Each usage of \texttt{\copy\mypic} puts only a pointer to the picture data in the PDF.

If you want to copy the same picture in different sizes, then choose a “basic size” used in \texttt{\setbox} and all different sizes can be realized by the \texttt{\transformbox}{\langle \textit{transformation} \rangle}{\copy\mypic}.

\texttt{\inspic} accepts old syntax \texttt{\inspic \langle filename \rangle <space>} or new syntax \texttt{\inspic\{\langle filename \rangle\}}. So, we need to define two auxiliary macros \texttt{\_inspicA} and \texttt{\_inspicB}.

You can include more \texttt{\pdfximage} parameters (like \texttt{page\langle number \rangle}) in the \texttt{\_picparams} macro.

All \texttt{\inspic} macros are surrounded in \texttt{\hbox} in order user can write \texttt{\moveright\inspic ...} or something similar.

Inkscape can save a picture to *.pdf file and labels for the picture to *.pdf_tex file. The second file is in \LaTeX{} format (unfortunately) and it is intended to read immediately after *.pdf is included in order to place labels of this picture in the same font as the document is printed. We need to read this \LaTeX{} file by plain \TeX{} macros when \texttt{\inkinspic} is used. These macros are stored in the \texttt{\_inkdefs} tokens list and it is used locally in the group. The solution is borrowed from OPmac trick 0032.
\pdfscale{\txt{\(x\)-scale}}\{\txt{\(y\)-scale}} and \pdfrotate{\txt{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 \transformbox{\{\txt{transformation}\}}\{\txt{\(text\)}} is copied from OPmac trick 0046. The \rotbox{\txt{\(degrees\)}}\{\txt{\(text\)}} is a combination of \rotsimple from OPmac trick 0101 and the \transformbox. Note, that \rotbox{-90} puts the rotated text to the height of the outer box (depth is zero) because code from \rotsimple is processed. But \rotbox{-90.0} puts the rotated text to the depth of the outer box (height is zero) because \transformbox is processed.
\_scantwodimens scans two objects with the syntactic rule \langle \text{dimen} \rangle and returns \{\langle \text{number} \rangle\} \{\langle \text{number} \rangle\} in sp unit.

\texttt{\textbackslash puttext} \langle \text{right} \rangle \langle \text{up} \rangle \{\langle \text{text} \rangle\} puts the \langle \text{text} \rangle to desired place: From current point moves \langle \text{down} \rangle and \langle \text{right} \rangle, puts the \langle \text{text} \rangle and returns back. The current point is unchanged after this macro ends.

\texttt{\textbackslash putpic} \langle \text{right} \rangle \langle \text{up} \rangle \langle \text{width} \rangle \langle \text{height} \rangle \{\langle \text{image-file} \rangle\} does \texttt{\textbackslash puttext} with the image scaled to desired \langle \text{width} \rangle and \langle \text{height} \rangle. If \langle \text{with} \rangle or \langle \text{height} \rangle is zero, natural dimension is used. The \texttt{\textbackslash nospec} is a shortcut to such a natural dimension.

\texttt{\textbackslash backgroundpic} \{\langle \text{image-file} \rangle\} puts the image to the background of each page. It is used in the \texttt{\textbackslash slides} style, for example.

\begin{verbatim}
\_def\_scantwodimens{%
   \directlua{tex.print(string.format('{\_pcent d}{\_pcent d}',
      token.scan_dimen(),token.scan_dimen()))}%
}
\_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(#3)}\_pgbackground=\_copy\_bgbox
\_pgbackground\_nospec=\_nopage\_nospec=0pt
\_public \puttext \putpic \_backgroundpic ;
\end{verbatim}

\_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 \texttt{\pdfliteral} argument.
The \texttt{\_oval\{\text{\texttt{text}}\}} is an example of \texttt{\_oval} usage.
The \texttt{\_incircle\{\text{\texttt{text}}\}} is an example of \texttt{\_circle} usage.
The \texttt{\_ratio}, \texttt{\_lwidth}, \texttt{\_fcolor}, \texttt{\_lcolor}, \texttt{\_shadow} and \texttt{\_overlapmargins} are parameters, they can be set by user in optional brackets \[\ldots\]. For example \texttt{\_fcolor=\Red} does \texttt{\_let\_fcolorvalue=\Red} and it means filling color.
The \texttt{\_setflcolor} uses the \texttt{\_fillstroke} macro to separate filling color and drawing color.
A shadow effect is implemented here. The shadow is equal to the silhouette of the given path in a gray-transparent color shifted by \shadowmoveto vector and with blurred boundary. A waistline with the width 2\shadowb around the boundary is blurred. The \shadowlevels levels of transparent shapes is used for creating this effect. The \shadowlevels+1/2 level is equal to the shifted given path.

The \pdfpageresources primitive is used to define transparency. It does not work when used in a box. So, we use it at the beginning of the output routine. The modification of the output routine is done using \insertshadowresources only once when the shadow effect is used first.

The \doshadow\{⟨\curve⟩\} does the shadow effect.
A generic macro `\clipinpath{x}{y}{curve}{text}` declares a clipping path by the `{curve}` shifted by the `{x}, {y}`. The `{text}` is typeset when such clipping path is active. Dimensions are given by bp without the unit here. The macros `\clipinoval{x}{y}{width}{height}{text}` and `\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 `{declaration}` part of `\table({declaration}){data}` includes column declarators (letters) and other material: the | or `(cmd)`. If the boundary declarator : is not used then the boundaries of columns are just before each column declarator with exception of the first one. For example, the declaration `{|c||c(xx)(yy)c}` should be written more exactly using the boundary declarator : `{|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 `(cmd)` are parts of the given table item (depending on the boundary declarator position). If you want to apply a special setting for a given column, you can do this by `(setting)` followed by column declarator. But if the column is not first, you must use `(setting)`. For example, we have three centered columns, the second one have to be in bold font and the third one have to be in red: `\table{c:(\bf)c:(\Red)c}{data}`

2.30.2 Usage of the `\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, \texttt{\tabskip} is 0pt. It means that only \texttt{\tabiteml}, \texttt{\tabitemr} and \texttt{(⟨cmds⟩)} can generate visual spaces between columns. But they are not real spaces between columns because they are in fact the part of the total column width.

The \texttt{\tabskip} value declared before the \texttt{\table} macro (or in \texttt{\everytable} or in \texttt{\thistable}) is used between all columns in the table. This value is equal to all spaces between columns. But you can set each such space individually if you use \texttt{⟨cmds⟩} in the \texttt{⟨declaration⟩} immediately before boundary character. The boundary character represents the column pair for which the \texttt{\tabskip} has individual value. For example \texttt{c(\tabskip=5pt):r} gives \texttt{\tabskip} value between \texttt{c} and \texttt{r} columns. You need not use boundary character explicitly, so \texttt{c(\tabskip=5pt)r} gives the same result.

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

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

### 2.30.3 Tables to given width

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

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

Example of \texttt{\table to⟨size⟩}:

\begin{verbatim}
\thistable{\tabskip=0pt plus1fil minus1fil}
\table to\hsize {lr}{⟨data⟩}
\end{verbatim}

This table has its width \texttt{\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 \texttt{\hsize}.

Example of \texttt{\table pxto⟨size⟩} (means “paragraphs expanded to”):

\begin{verbatim}
\table pxto\hsize {c|p{\tsize}|}{\crl
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. \crl
bb ddd ggg & Dsjds ds cgha sfgs dd fddzf dfhz xxz
ddkas jd dsjds ds cgha sfgs dd fddzf. \crl
}
\end{verbatim}

| aaaa & 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. \crl |
| bb ddd ggg & Dsjds ds cgha sfgs dd fddzf dfhz xxz
ddkas jd dsjds ds cgha sfgs dd fddzf. \crl |

The first \texttt{c} column is variable width (it gets the width of the most wide item) and the resting space to given \texttt{\hsize} is filled by the \texttt{p} column.

You can declare more than one \texttt{p} columns in the table when \texttt{pxto} keyword is used. The total sum of \texttt{⟨coefficients⟩\tsize} must be exactly one. For example,

\begin{verbatim}
\table pxto13cm {r p{.3\tsize} p{.5\tsize} p{.2\tsize} l}{⟨data⟩}
\end{verbatim}

This gives the ratio of widths of individual paragraphs in the table.

### 2.30.4 \texttt{\eqbox}: boxes with equal width across the whole document

The \texttt{\eqbox ⟨⟨label⟩⟩⟨⟨text⟩⟩} behaves like \texttt{\bbox⟨⟨text⟩⟩} in the first run of \texttt{\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 \texttt{\TeX} run. Then \texttt{\eqbox ⟨⟨label⟩⟩⟨⟨text⟩⟩} behaves like \texttt{\bbox to ⟨dim:label⟩ \hspace{⟨text⟩}} where \texttt{⟨dim:label⟩} is the maximum width of all boxes labeled...
by the same \([\langle \text{label} \rangle]\). The documentation of the \LaTeX{} package \texttt{eqnarray} includes more information and tips.

The \texttt{eqbox} \([\langle \text{dim} \rangle\{\langle \text{label} \rangle\}]\{\langle \dimen \rangle\}\) expands to \(\langle \text{dim} \rangle:\langle \text{label} \rangle\) if this value is known, else it expands to the given \(\langle \dimen \rangle\).

The optional parameter \texttt{r} or \texttt{l} can be written before \([\langle \text{label} \rangle\}]\{\langle \dimen \rangle\}\) if you want to put the text to the right or to the left side of the box width.

Try the following example and watch what happens after first \TeX{} run and after the second one.

\begin{verbatim}
def\leftitem#1{\par
\noindent \hangindent=\eqboxsize[items]{2em}\hangafter=1
\eqbox r[items]{#1 }\ignorespaces}
def\leftitem {f first} \lorem[1]
def\leftitem {f second one} \lorem[2]
def\leftitem {f final} \lorem[3]
\end{verbatim}

2.30.5 Implementation of the \texttt{\table} macro and friends

The result of the \texttt{\table}\{\langle \text{declaration} \rangle\}\{\langle \text{data} \rangle\} macro is inserted into \texttt{\_tablebox}. You can change default value if you want by \texttt{\let\_tablebox=\vtop} or \texttt{\let\_tablebox=\relax}.

We save the \texttt{to(size)} or \texttt{pxto(size)} to \#1 and \texttt{\_tableW} sets the \texttt{to(size)} to the \texttt{\_tableW} macro. If \texttt{pxto(size)} is used then \texttt{\_tableW} is empty and \texttt{\_tmpdim} includes given \(\langle \text{size} \rangle\). The \texttt{\_ifpxto} returns true in this case.

The \texttt{\table} continues by reading \{\langle \text{declaration} \rangle\} in the \texttt{\_tableA} macro. Catcodes (for example the \texttt{|} character) have to be normal when reading \texttt{\table} parameters. This is the reason why we use \texttt{\catcode\table} here.

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

The \texttt{\_tableA\{\langle \text{declaration} \rangle\}} macro scans the \langle \text{declaration} \rangle by \texttt{\_scantabdata\#1} and \texttt{\_tabdata={\_colnum\_relax}\_scantabdata\#1} and \texttt{\_the\_everytable\_bgroup \_catcode\#=12 \_tableB}. The trick \texttt{\_tmptoks={\{\langle \text{data} \rangle\}}\_edef\_tmpb{\_the\_tmptoks} is used here in order to keep the hash marks in the \langle \text{data} \rangle unchanged.

The \texttt{\tableB} saves \langle \text{data} \rangle to \texttt{\_tmpp} and does four \texttt{\replstring}s to prefix each macro \texttt{\_crl} (etc.) by \texttt{\_crr}. The reason is: we want to use macros that scan its parameter to the delimiter written in the
right part of the table item declaration. See \fS for example. The \crcr cannot be hidden in another macro in this case.

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

\tabskip=\tabskipl \halign{⟨\converted declaration⟩}\tabskip=\tabskipr \cr

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

If pxto keyword was used, then we set the virtual unit \tsize to \hsize first. Then the first attempt of the table is created in box 0. Then the \tsize is re-calculated using \wd0 and the real table is printed by \halign in the second pass.

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

Because the color selector with \aftergroup can be used inside the table item, we must create the second real group for each table item. This is reason why we start \{converted declaration\} by \bgroup and we end it by \egroup in \_tableC macro. Each & character is stored as \egroup&\bgroup in \⟨\converted declaration⟩. The \halign\_tablew\_tableC really does:

\halign\_tablew\{\bgroup \⟨\converted declaration⟩ \egroup \tabskip=\tabskipr \cr \langle \data \cr \}

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

\_long\_def\_tableB #1\{\_def\_tmpb{#1}\
\_replstring\_tmpb{\crl}{\crcr\crl} \_replstring\_tmpb{\crll}{\crcr\crll}\
\_replstring\_tmpb{\crli}{\crcr\crli} \_replstring\_tmpb{\crlli}{\crcr\crlli}\
\_edef\_tabskipmid{\_the\_tabskip} \_tabskip=\_tabskipl \_ifpxto\
\_tsize=\_hsize \_setbox0 = \_vbox\{\_tablepxpreset \_halign \_tableC}\_tsize=\_dimexpr\_hsize-(\wd0-\_tmpdim)\relax\
\_setbox0=\_null \_halign \_tableC\_else\_halign\_tablew \_tableC\_fi \_egroup

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 \langle declaration⟩=|cr||cl|.

\_vrule\_the\_tabiteml\_hfil\_unsskip\_hfil\_the\_tabitemr\_tabstrutA
\_the\_tabiteml\_hfil\_unsskip\_the\_tabitemr
\_vrule\_kern\_vkkern\_vrule\_tabstrutA
\_the\_tabiteml\_hfil\_unsskip\_hfil\_the\_tabitemr\_tabstrutA
\_the\_tabiteml\_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 \textit{converted declaration}. This code is \texttt{\egroup &\bgroup \colnum=} \texttt{⟨value⟩} \texttt{.} You can get the current number of column from the \texttt{\colnum} register, but you cannot write \texttt{\the\colnum} as the first object in a \texttt{⟨data⟩} item because \texttt{\halign} first expands the front of the item and the left part of the declaration is processed after this. Use \texttt{\relax\the\colnum} instead. Or you can write:

\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 \_addtabvrule adds the \_addtable data (used between columns) to the \textit{converted declaration}. This code converts \texttt{||} or \texttt{|} from \texttt{\table ⟨declaration⟩} to the \textit{converted declaration}.

\begin{verbatim}
\def\tabdeclarec{⟨\the\tabiteml\hfil##\unsskip\hfil\the\tabitemr⟩}
\def\tabdeclarel{⟨\the\tabiteml\relax##\unsskip\hfil\the\tabitemr⟩}
\def\tabdeclarer{⟨\the\tabiteml\hfil##\unsskip\the\tabitemr⟩}
\def\paramtabdeclarep#1{⟨\addtabelitem\def\addtabelitem{\let\addtabelitem=\addtabelitemx}\addtabitem{\vtop\{\hsize=#1\relax\baselineskip=\normalbaselineskip\lineskiplimit=\zo\noindent\hskip\zo\relax##\unsskip\ifvmode\vskip\dp\tstrutbox\else\lower\dp\tstrutbox\hbox{}}\fi\the\tabitemr⟩}
\end{verbatim}

Users put optional spaces around the table item typically, i.e. they write \texttt{& text} & instead \texttt{&text&}. The left space is ignored by the internal \texttt{\TeX} algorithm but the right space must be removed by macros. This is a reason why we recommend to use \texttt{\unskip} after each \texttt{##} in your definition of “declaration letters”. This macro isn’t only the primitive \texttt{\unskip} after each \texttt{##} because we allow usage of plain \texttt{\TeX} \texttt{\hideskip} macro: \texttt{&\hideskip text\hideskip&}.
The \fL, \fR, \fC and \fX macros only do special parameters settings for paragraph building algorithm. The \fS prints the paragraph into box 0 first, measures the number of lines by the \prevgraf primitive and use (or don't use) \hfil (for centering) before the first line.

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\{number\}\{declaration\}\{text\} macro generates similar \omit\span\omit\span sequence as plain \TeX macro \multispan. Moreover, it uses \scantabdata to convert \{declaration\} from \table syntax to \halign syntax.

The \vspan\{number\} \{text\} implementation is here. We need to lower the box by \((\text{number}-1)\times(\text{ht}+\text{dp of \tabstrut}) / 2\). The \#1 parameter must be a one-digit number. If you want to set more digits then use braces.
The parameters of primitive \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.

\begin{verbatim}
\newdimen\drulewidth \drulewidth=0.4pt
\let\orihrule=\hrule \let\orivrule=\vrule
\def\rulewidth{\afterassignment\rulewidthA \drulewidth}
\def\rulewidthA{\edef\hrule{\orihrule height\drulewidth}%
\edef\vrule{\orivrule width\drulewidth}%
\let\rulewidth=\drulewidth}
\public \vrule \hrule \rulewidth; \end{verbatim}

The \frame{⟨text⟩} uses “\vbox in \vtop” trick in order to keep the baseline of the internal text at the same level as outer baseline. User can write \frame{abcxyz} in normal paragraph line, for example and gets the expected result: abcxyz. The internal margins are set by \vvkern and \hhkern parameters.

\begin{verbatim}
\longdef\frame#1{%
hbox{\vrule vtop{\vbox{\hrule \kern \vvkern
\hbox{\kern \hhkern \relax#1 \kern \hhkern}}%\kern \vvkern \hrule}}
\public \frame ; \end{verbatim}

\eqbox and \eqboxsize are implemented here. The widths of all \eqboxes are saved to the .ref file in the format \Xeqbox{⟨label⟩}{⟨size⟩}. The .ref file is read again and maximum box width for each \label is saved to \eqb{⟨label⟩}.

\begin{verbatim}
\def\Xeqbox#1#2{%\ifcsname \eqb:#1\endcsname\ifdim #2>\cs{\eqb:#1}\relax \sdef{\eqb:#1}{#2}\fi\else \sdef{\eqb:#1}{#2}\fi}
\def\eqbox #1[#2]#3{\setbox0=\hbox{{#3}}%\openref \immediate \wref \Xeqbox{{#2}{\the\wd0}}%\ifcsname \eqb:#2\endcsname\hbox to\cs{\eqb:#2}{\ifx r#1\hfill\fi\hss\unhbox0\hss\ifx l#1\hfill\fi}\else \box0 \fi\else \box0 \fi}
\def\eqboxsize [#1]#2{\trycs{\eqb:#1}{#2}}\public \eqbox \eqboxsize ; \end{verbatim}

### 2.31 Balanced multi-columns

This code is documented in detail in the “\TeXbook” pages 244–246, free available, \url{http://petr.olsak.net/tbn.html}, but in Czech. Roughly speaking, macros complete all material between \begmulti⟨num-columns⟩ and \endmulti into one \vbox 6. Then the macro measures the amount of free space at the current page using \pagegoal and \pagtotal and does \vsplit of \vbox 6 to columns with a height of such free space. This is done only if we have enough amount of material in \vbox 6 to fill the full page by columns. This is repeated in a loop until we have less amount of material in \vbox 6. Then we run \balancecolumns which balances the last part of the columns. Each part of printed material is distributed to the main vertical list as \hbox{⟨columns⟩} and we need not do any change in the output routine.

If you have paragraphs in \begmulti... \endmulti environment then you may say \raggedright inside this environment and you can re-assign \widowpenalty and \clubpenalty (they are set to 10000 in Op\TeX).
Splitting columns...

Final balancing of the columns.
2.32 Citations, bibliography

2.32.1 Macros for citations and bibliography preloaded in the format

\citetext{cite-bib.opm}

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.

\_def\bibnum \newcount % the bibitem counter
\def\bibmark \newtoks % the bibmark used if \nonumcitations
\def\lastcitenum \newcount % for \shortcitations
\def\bibmark \newtoks
\def\bibitem \newcount % the bibitem counter

\bib expands to \bibpart/. By default, \bibpart is empty, so internal links are in the form \cite:\(\text{number}\). If \bibpart is set to \bibpart/\(\text{number}\), then internal links are \cite:\bibpart/\(\text{number}\).

\citetext{cite-bib.opm}

\\cite \bibpart/\(\text{label}\),...\bibpart/\(\text{label}\)\] manages \texttt{\{labels\}} using \\cite\texttt{\[\]} and prints \bibpart/\(\text{mark}\) using \\printsavedcites. \\nocite \bibpart/\(\text{label}\),...\bibpart/\(\text{label}\)\] only manages \texttt{\{labels\}} but prints nothing. \\rcite \bibpart/\(\text{label}\),...\bibpart/\(\text{label}\)\] behaves like \\cite but prints \bibpart/\(\text{mark}\) without brackets. \\ecite \bibpart/\(\text{label}\),...\bibpart/\(\text{label}\)\] behaves like \\rcite \bibpart/\(\text{label}\) but prints \texttt{\textit{\{text\}}} instead \bibpart/\(\text{mark}\). The \texttt{\textit{\{text\}}} is hyperlinked like \bibpart/\(\text{bib}-\text{mark}\) when \cite or \rcite is used. The empty internal macro \\savedcites will include the \bibpart/\(\text{bib}-\text{mark}\) list to be printed. This list is set by \\cite\texttt{\[\]} inside a group and it is used by \\printsavedcites in the same group. Each \\cite/\rcite/\ecite macro starts from empty list of \bibpart/\(\text{mark}\) because new group is opened.

\citetext{cite-bib.opm}

\\citex \texttt{\{label\}},...\texttt{\{label\}}\] processes one label from the list of labels given in the parameter of \\cite, \\nocite, \\rcite or \\ecite macros. It adds the \texttt{\{label\}} to a global list \\ctlst:\bibpart/ which will be used by \\usebib (it must know what \texttt{\{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 \texttt{\{label\}} in the \\ctlst:\bibpart/, we distinguish four cases:
• \(\langle\text{label}\rangle\) was not declared by \texttt{\_Xbib} before and it is first such a \(\langle\text{label}\rangle\) in the document: Then \texttt{\_bib:(\textit{bibpart})/\langle\text{label}\rangle}\) is undefined and we save label using \texttt{\_addcitelist}, write warning on the terminal and define \texttt{\_bib:(\textit{bibpart})/\langle\text{label}\rangle}\) as empty.

• \(\langle\text{label}\rangle\) was not declared by \texttt{\_Xbib} before but it was used previously in the document: Then \texttt{\_bib:(\textit{bibpart})/\langle\text{label}\rangle}\) is empty and we do nothing (only data to \texttt{\_savedcites} are saved).

• \(\langle\text{label}\rangle\) was declared by \texttt{\_Xbib} before and it is first such \(\langle\text{label}\rangle\) used in the document: Then \texttt{\_bib:(\textit{bibpart})/\langle\text{label}\rangle}\) includes \texttt{\_bibnn\{\langle\text{number}\rangle\}}\) and we test this case by the command \texttt{\if\&\_bibnn\{\langle\text{number}\rangle\}}\) is true when \texttt{\_bibnn\{\langle\text{number}\rangle\}}\) expands to empty. The \(\langle\text{label}\rangle\) is saved by \texttt{\_addcitelist} and \texttt{\_bib:(\textit{bibpart})/\langle\text{label}\rangle}\) is re-defined directly as \texttt{\langle\text{number}\rangle}.

• \(\langle\text{label}\rangle\) was declared by \texttt{\_Xbib} and it was used previously in the document. Then we do nothing (only data to \texttt{\_savedcites} are saved.

The \texttt{\_citeA} macro runs repeatedly over the whole list of \(\langle\text{label}\rangle\).

Because we implement possibility of more independent bibliography lists distinguished by \texttt{\langle\textit{bibpart}\rangle}, the \texttt{\_addcitelist\{\langle\text{label}\rangle\}} macro must add the \(\langle\text{label}\rangle\) to given \texttt{\_ctlst:}\texttt{\langle\text{bibpart}\rangle}/. When \texttt{\_addcitelist} is processed before \texttt{\usebib}, then \texttt{\_citeI\{\langle\text{label}\rangle\}}\) is added. \texttt{\usebib} will use this list for selecting right records from \texttt{.bib} file. Then \texttt{\usebib} sets \texttt{\_ctlst:}\texttt{\langle\text{bibpart}\rangle}/ to \texttt{\_write}.

If \texttt{\_addcitelist} is processed after \texttt{\usebib}, then \texttt{\_Xcite\{\langle\text{bibpart}\rangle\}/\{\langle\text{label}\rangle\}}\) is saved to the \texttt{.ref} file. The \texttt{\_Xcite} creates \texttt{\_ctlstB:\{\langle\text{bibpart}\rangle\}/ as a list of saved \texttt{\_citeI\{\langle\text{label}\rangle\}}. Finally, \texttt{\usebib} concatenates both lists \texttt{\_ctlst:}\texttt{\langle\text{bibpart}\rangle}/ and \texttt{\_ctlstB:}\texttt{\langle\text{bibpart}\rangle}\) in the second \TeX run.

The \texttt{\langle\text{bib-marks}\rangle} (in numeric or text form) are saved in \texttt{\_savedcites} macro separated by commas. The \texttt{\_printsavedcites} prints them by normal order or sorted if \texttt{\sortcitations} is specified or condensed if \texttt{\shorthortcitations} is specified.

The \texttt{\sortcitations} appends the dummy number 300000 and we suppose that normal numbers of bib-entries are less than this constant. This constant is removed after the sorting algorithm. The \texttt{\shorthortcitations} sets simply \texttt{\_lastcitenum}=1. The macros for \texttt{\langle\text{bib-marks}\rangle} printing follows (sorry, without detail documentation). They are documented in \texttt{opmac-d.pdf} (but only in Czech).
The \bib{[label]} or \bib{(label)} =\{\bibmark\} prints one bib-entry without reading any database. The bib-entry follows after this command. This command counts the used \bibnum counter and saves \_Xbib{⟨bibpart⟩}{⟨label⟩}{⟨number⟩}{⟨nonumber⟩} into .ref file immediately using \_wbib{⟨label⟩}{⟨number⟩}{⟨nonumber⟩}. This is the core of creation of mapping from ⟨labels⟩ to ⟨number⟩ and ⟨nonumber⟩. \_bibA and \_bibB implement the scanner of the optional argument with the \_bibmark.

\_bibgl is \relax by default but \slides do \let\_bibgl=\_global.

\_dbib{⟨label⟩} creates destination for hyperlinks.
The \printbib prints the bib-entry itself. You can re-define it if you want a different design. The \printbib starts in horizontal mode after \noindent and after the eventual hyperlink destination is inserted. By default, the \printbib sets the indentation by \hangindent and prints numeric \bibmarks by \llap{[the\bibnum]}. If \nonumcitations then the \citelinkA is not empty and \bibmarks \bibnum nor \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 \bib-labels declares a list of bib labels which are not fully declared in \bib file but we want to suppress the warning about it. List of bib labels are comma-separated case sensitive list without spaces.

2.32.2 The \usebib command

The file \usebib.opm implements the command \usebib\sorttype \style \bibfiles where \sorttype is one letter c (references ordered by citation order in the text) or s (references ordered by key in the style file), \style is the part of the name \bib\langle\style\rangle.opm of the style file and \bibfiles are one or more \bib files names without suffix separated by comma without space. Example:

\usebib/s (simple) mybase,yourbase

This command reads the \bibfiles directly and creates the list of bibliographic references (only those declared by \cite or \nocite in the text). The formatting of such references is defined in the style file. The principle “first entry wins” is used. Suppose \usebib/s (simple) local,global. If an entry with the same label is declared in local.bib and in global.bib too then the first wins. So, you can set exceptions in your local.bib file for your document.

The bib-(\style).opm declares entry types (like @BOOK, @ARTICLE) and declares their mandatory and optional fields (like author, title). When a mandatory field is missing in an entry in the \bib file then a warning is printed on the terminal about it. You can suppress such warnings by command \nobibwarning \bib-labels, where \bib-labels is a comma-separated list of labels (without spaces) where missing mandatory fields will be no warned.

Old \bib files may use the obscure notation for accents like \"ö\". Recommendation: convert such old files to Unicode encoding. If you are unable to do this then you can set \bibtexhook={\oldaccents}. 
2.32.3 Notes for bib-style writers

The .bib files include records in the format:

```
\@{entry-type}\{\{label\},
\{field-name\} = "\{field-data\}",
\{field-name\} = "\{field-data\}",
...etc
}
```

see the file demo/op-biblist.bib for a real example. The \{entry-types\} and \{field-names\} are case insensitive.

Ancient Bib\TeX{} has read such files and has generated files appropriate for reading by \LaTeX{}. 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.omp and bib-iso690.omp. 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\}.omp 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\}.
- The part \{if not defined\} can include the \_\_bibwarning command if the \{field-name\} is mandatory.
- \_\_bprintb ([field-name]) \{if defined\} \{if not defined\}. The same as \_\_bprinta, but the \#\#1 parameter is used instead *. Differences: \#\#1 parameter can be used more than once and can be enclosed in nested braces. The * parameter can be used at most once and cannot be enclosed in braces. Warning: if the \_\_bprintb commands are nested (\_\_bprintb in \_\_bprintb), then you need to write the \#\#\#\#1 parameter for internal \_\_bprintb. But if \_\_bprinta commands are nested then the parameter is not duplicated.
- \_\_bprintc \_\_macro \{if non-empty\}. The \{if non-empty\} part is executed if \_\_macro is non-empty. The * parameter can be used, it is replaced by the \_\_macro.
- \_\_bprintv ([\{field1\}, \{field2\}, ...]) \{if defined\} \{if not defined\}. The part \{if defined\} is executed if \{field1\} or \{field2\} or ... is defined, else the second part \{if not defined\} is executed. There is one field name or the list field names separated by commas. The parts cannot include any parameters.

There are two special field-names: \_\_author and \_\_editor. The processed list of authors or editors are printed here instead of raw data, see the commands \_\_authorname and \_\_editorname below.

The bib-style writer can define \_\_print:BEGIN and/or \_\_print:END. They are executed at the beginning or end of each \{entry-type\}. The formatting does not solve the numbering and paragraph indentation of the entry. This is processed by \_\_printbib macro used in Op\TeX{} (and may be redefined by the author or document designer).

The \_\_bibmark={something} can be declared, for instance in the \_\_print:END macro. Such “bibmark” is saved to the .ref file and used in next \TeX{} run as \cite marks when \nonumcitations is set.

Moreover, the bib-style writer must declare the format of special fields \_\_author and \_\_editor. These fields include a list of names, each name is precessed individually in a loop. The \_\_authorname or \_\_editorname is called for each name on the list. The bib-style writer must define the \_\_authorname and \_\_editorname commands in order to declare the format of printing each individual name. The following control sequences can be used in these macros:

- \_\_NameCount: the number of the currently processed author in the list
- \_\_namecont: the total number of the authors in the list
- \_\_Lastname, \_\_Firstname, \_\_Von, \_\_Junior: the parts of the name.
The whole style file is read in the group during the \usebib command is executed before typesetting the reference list. Each definition or setting is local here.

The auto-generated phrases (dependent on current language) can be used in bib-style files by \texttt{\_mtext{bib.\langle identifier\rangle}}; where \langle identifier\rangle is an identifier of the phrase and the phrase itself is defined by \texttt{\_sdef\_mt:bib.\langle identifier\rangle: \langle language\rangle\{\langle phrase\rangle\}}. See section \ref{sec:2.37.3} for more detail. Phrases for \langle identifiers\rangle: and, etal, edition, citedate, volume, number, prepages, postpages, editor, editors, available, availablealso, baechthesis, masthesis, phdthesis are defined already, see the end of section \ref{sec:2.37.3}.

If you are using non-standard field-names in .bib database and bib-style, you have to declare them by \texttt{\_CreateField \{\langle fieldname\rangle\}}.

You can declare \texttt{\_SortingOrder} in the manner documented by librarian package.

User or author of the bib-style can create the hidden field which has a precedence while sorting names. Example:

\begin{verbatim}
\CreateField \{sortedby\}
\SpecialSort \{sortedby\}
\end{verbatim}

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 \texttt{sortedby = "\&"}, because this character is sorted before A.

\subsection{The usebib.omp macro file loaded when \usebib is used}

Loading the librarian.tex macro package. See \texttt{texdoc librarian} for more information about it.

We want to ignore \texttt{\_errmessage} macro and we want not to create \texttt{\jobname.lbr} file.

The \texttt{\usebib} command.

\begin{verbatim}
\def\_usebib/#1 (#2) #3 {\%\n  \let\_citeI=\_relax \xdef\_citelist\{\_trycs{\_ctlst:\_bibp}{}{\_ctlstB:\_bibp}{}}\%\n  \global \_ea \_let \_csname _ctlst:\_bibp\_endcsname =\_write \_ifx\_citelist\_empty\_else % there was \nocite\{\_relax\} used.\n  \_opwarning{No cited items. \_noexpand\usebib ignored}\%\n  \_else\%\n    \_bgroup \_par\%\n      \_emergencystretch=\_3\hspace\%\n      \def\_optexpbibstyle\{\_2\}\%
      \setctable\_optexpcatcodes\%\n      \ea \_skiptoendinput \_input languages.omp\%\n      \_input bib-\_2.omp \%\n      \the \_bibtexhook\%\n      \_ifcsname _mt:bib.\_and\_cs{\_lan: \_the\_language}\_endcsname \_else\%
        \_opwarning{\_string\usebib: No phrases for language \_cs{\_lan: \_the\_language} (using \"en\")\%\n          \_language=\_\_chardef\_documentlanguage=\_0\%
        \_fi\%
      \_fi\%
      \_def\_tmp\{\_relax\}\%
      \_if\_tmp\_empty\_else % there was \nocite\{\_relax\} used.
\end{verbatim}

154
Corrections in librarian macros.

Main action per each entry.
Various macros + multilingual. Note that \nobibwarnlist is used in \bibwarning and it is set by \nobibwarning macro.
2.32.5 Usage of the bib-iso690 style

This is the iso690 bibliographic style used by OpTEX.

See op-biblist.bib for an example of the .bib input. You can try it by:

\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 = \textit{value}, or name = \texttt{value}. No matter which form is used. If the value is pure numeric then
you can say simply name = value. Warning: the comma after each field value is mandatory! If it is
missing then the next field is ignored or badly interpreted.

The entry names and field names are case insensitive. If there exists a data field no mentioned here
then it is simply ignored. You can use it to store more information (abstract, for example).

There are “standard fields” used in ancient bibTEX (author, title, editor, edition, etc., see
http://en.wikipedia.org/wiki/BibTeX). The iso690 style introduces several “non-standard” fields: ednote,
numbering, isbn, issn, doi, url, citedate, key, bibmark. They are documented here.

Moreover, there are two optional special fields:

- lang = language of the entry. The hyphenation plus autogenerated phrases and abbreviations will
be typeset by this language.
- option = options by which you can control a special printing of various fields.

There can be only one option field per each entry with (maybe) more options separated by spaces.
You can declare the global option(s) in your document applied for each entry by \biboptions={...}.

The author field
All names in the author list have to be separated by “ \textit{and} ”. Each author can be written in various
formats (the \textit{von} part is typically missing):

Firstname(s) von Lastname
or
von Lastname, Firstname(s)
or
von Lastname, After, Firstname(s)

Only the Lastname part is mandatory. Examples:

Petr Olšák
or
Olšák, Petr

Leonardo Piero da Vinci
or
da Vinci, Leonardo Piero
or
da Vinci, painter, Leonardo Piero

The separator “ \textit{and} ” between authors will be converted to comma during printing, but between the
semifinal and final author the word “and” (or something different depending on the current language) is
printed.

The first author is printed in reverse order: “LASTNAME, Firstname(s) von, After” and the other
authors are printed in normal order: “Firstname(s) von LASTNAME, After”. This feature follows the
ISO 690 norm. The Lastname is capitalized using uppercase letters. But if the \caps font modifier is
defined, then it is used and printed {\caps\textit{rm} Lastname}. 

157
You can specify the option \texttt{aumax}:(\texttt{number}). The \texttt{number} denotes the maximum authors to be printed. The rest of the authors are ignored and the \texttt{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 \texttt{.bib} file as you need to print but you want to append \texttt{et-al.} then you can use \texttt{auetal} option.

There is an \texttt{aumin}:(\texttt{number}) 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}:(\texttt{number}) option isn’t given. There is no internal limit. But you can set the global options in your document by setting the \texttt{\biboptions} tokens list. For example:

\begin{verbatim}
\biboptions={aumax:7 aumin:1}
\end{verbatim}

% if there are 8 or more authors then only the first author is printed.

Examples:

\begin{verbatim}
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.
\end{verbatim}

If you need to add a text before or after the author’s list, you can use the \texttt{auprint}:\{\texttt{value}\} option. The \texttt{value} will be printed instead of the authors list. The \texttt{value} can include \texttt{\AU} macro which expands to the authors list. Example:

\begin{verbatim}
author = "Robert Calbraith",
option = "auprint:{\\AU\space [pseudonym of J. K. Rowling]}"
output: CALBRAITH Robert [pseudonym of J. K. Rowling].
\end{verbatim}

You can use the \texttt{autrim}:(\texttt{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 \texttt{number}. There is an exception: \texttt{autrim:0} means that no Firstnames are trimmed. This is the default behavior. Another example: \texttt{autrim:1} means that all Firstnames are trimmed.

\begin{verbatim}
author = "John Green and Bob Brown and Alice Black",
option = "auetal autrim:1",
output: GREEN, J., B. BROWN, A. BLACK et al.
\end{verbatim}

If you need to write a team name or institution instead of authors, replace all spaces by \texttt{\cdot} in this name. Such text is interpreted as Lastname. You can add the secondary name (interpreted as Firstname) after the comma. Example:

\begin{verbatim}
author = "Czech\ Technical\ University\ in\ Prague, Faculty\ of\ Electrical\ Engineering",
output: CZECH TECHNICAL UNIVERSITY IN PRAGUE, Faculty of Electrical Engineering.
\end{verbatim}

The editor field
The editor field is used for the list of the authors of the collection. The analogous rules as in author field are used here. It means that the authors are separated by “ and ”, the Firstnames, Lastnames, etc. are
interpreted and you can use the options edmax:⟨number⟩, edmin:⟨number⟩, edetal, edtrim:⟨number⟩ and edprint:{⟨value⟩} (with \ED macro). Example:

editor = "Jan Tomek and Petr Karas",
option = "edprint:{\ED, editors.} edtrim:1",

Output: J. TOMEK and P. KARAS, editors.

If edprint option is not set then {\ED, eds.} or {\ED, ed.} is used depending on the entry language and on the singular or plural of the editor(s).

The ednote field
The ednote field is used as the secondary authors and more 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 ednote field 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$^\text{nd}$",
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",
option = "editionprint:{Second full revised edition}",


You can use \EDN macro in editionprint value. This macro is expanded to the edition value. Example:

edition = "Second",
option = "editionprint:{\EDN space full revised edition}",
or
edition = "Second full revised edition",
option = "editionprint:{\EDN}"

159
The address, publisher, year fields
This is an anachronism from ancient Bib\TeX{} (unfortunately no exclusive) that the address field includes only the city of the publisher’s residence. No more data are here. The publisher field includes the name of the publisher.

\begin{verbatim}
address = "Berlin",
publisher = "Springer Verlag",
year = 2012,
\end{verbatim}


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 \texttt{noautoletters} option.

You can use "\texttt{yearprint:\{\texttt{value}\}}" option. If it is set then the \texttt{(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:

\begin{verbatim}
year = 2000,
option = "yearprint:\{© 2000\}",
\end{verbatim}


\begin{verbatim}
year = "2012a",
option = "yearprint:\{2012\}",
\end{verbatim}


The address, publisher, and year are typically mandatory fields. If they are missing then the warning occurs. But you can set \texttt{unpublished} option. Then this warning is suppressed. There is no difference in the printed output.

The url field
Use it without \texttt{\url} macro, but with \texttt{http://} prefix. Example:

\begin{verbatim}
url = "http://petr.olsak.net/opmac.html",
\end{verbatim}

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 \url{http://petr.olsak.net/opmac.html}.

If the \texttt{cs} language is the current one than the output is:

Dostupné z: \url{http://petr.olsak.net/opmac.html}.

If the \texttt{urlalso} option is used, then the added text has the form “Available also from” or “Dostupné také z:” (if \texttt{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:

\begin{verbatim}
citedate = "2004/05/21",
\end{verbatim}

Output when \texttt{en} is current: [cit. 2004-05-21].

Output when \texttt{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:

\begin{verbatim}
howpublished = "online",
\end{verbatim}

Output: [online].

The volume, number, pages and numbering fields
The volume is the “big mark” of the journal issue and the number is the “small mark” of the journal issue and pages includes the page range of the cited article in the journal. The volume is prefixed by Vol. , the number by No. , and the pages by pp. . But these prefixes depends on the language of the entry.

Example:
volume = 31,
number = 3,
pages = "37--42",


volume = 31,
number = 3,
pages = "37--42",
lang = "cs",

Output: ročník 31, č. 3, s. 37–42.

If you disagree with the default prefixes, you can use the numbering field. When it is set then it is used instead of volume, number, pages fields and instead of any mentioned prefixes. The numbering can include macros \VOL, \NO, \PP, which are expanded to the respective values of fields. Example:

volume = 31,
number = 3,
pages = "37--42",

numbering = "Issue~\VOL/\NO, pages~\PP",

Output: Issue 31/3, pages 37–42

Note: The volume, numbers, and pages fields are printed without numbering field 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 Bib\TeX. But the ednote is mandatory field here, so you cannot use these entries from the old databases without warnings and without some additional work with the .bib file.

The cite-marks (bibmark) used when \nonumcitations is set
When \nonumcitations is set then \cite prints text-oriented bib-marks instead of numbers. This style file auto-generates these marks in the form “Lastname of the first author, comma, space, the year” if the bibmark field isn’t declared. If you need to set an exception from this common format, then you can use bibmark field.

The OPmac trick http://petr.olsak.net/OPmac-tricks-e.html#bibmark describes how to redefine the algorithm for bibmark auto-generating when you need the short form of the type [Au13].

Sorting
If \usebib/c is used then entries are sorted by citation order in the text. If \usebib/s is used then entries are sorted by “Lastname, Firstname(s)” of the first author and if more entries have this value equal, then the year is used (from older to newer). This feature follows the recommendation of the ISO 690 norm.

If you have the same authors and the same year, you can control the sorting by setting years like 2013, 2013a, 2013b, etc. You can print something different to the list using \yearprint\{\langle value \rangle\} 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:

\begin{verbatim}
author      = "Světla Čmejrková",
key        = "Czzmejrkova Svetla",
\end{verbatim}

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_{i=1}^n=", 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 \text{bib.\langle identifier\rangle} if you want to use a phrase dependent on outer document language (no on entry language). Example:

\begin{verbatim}
howpublished = "\text{bib.blue-ray}"
\end{verbatim}

Now, you can set the variants of bib.blue-ray phrase for various languages:
Summary of non-standard fields
This style uses the following fields unknown by bib\TeX:  
option ... options separated by spaces  
lang ... the language two-letter code of one entry  
ednote ... edition info (secondary authors etc.) or  
    global data in @MISC-like entries  
citedate ... the date of the citation in year/month/day format  
numbering ... format for volume, number, pages  
isbn ... ISBN  
issn ... ISSN  
doi ... DOI  
url ... URL

Summary of options  
aumax:⟨number⟩ ... maximum number of printed authors  
aumin:⟨number⟩ ... number of printed authors if aumax exceeds  
autrim:⟨number⟩ ... full Firstnames iff number of authors are less than this  
auprint:{⟨value⟩} ... text instead authors list (\textmacro{AU} macro may be used)  
edmax, edmin, edtrim ... similar as above for editors list  
edprint:{⟨value⟩} ... text instead editors list (\textmacro{ED} macro may be used)  
titlepost:{⟨value⟩} ... text after title  
yearprint:{⟨value⟩} ... text instead real year (\textmacro{YEAR} macro may be used)  
editionprint:{⟨value⟩} ... text instead of real edition (\textmacro{EDN} macro may be used)  
urlalso ... the 'available also from' is used instead 'available from'  
unpublished ... the publisher etc. fields are not mandatory  
nowarn ... no mandatory fields

Other options in the option field are silently ignored.

2.32.6 Implementation of the bib-iso690 style
Formating of Author/Editor lists.

\def\firstauthorformat{
  \upper{\Lastname}\_bprintc\_Firstname{, \_bprintc\_Von{ \_bprintc\_Junior{, \}}%}
}\def\otherauthorformat{
  \_bprintc\_Firstname{, \_bprintc\_Von{ \_bprintc\_Junior{, \}}}%}
\def\commonname{
  \ifnum\NameCount=1
    \firstauthorformat
  \else
    \ifx\dobibmark\undefined
      \edef\dobibmark{\Lastname}\fi
    \else
      \_ifx\maybeetal\empty \bibconjunctionand\_else \fi
    \else
      \_fi
    \_otherauthorformat
  \fi
}\def\authorname{
  \ifnum\NameCount>0\_namecount\relax\else \commonname \fi
  \ifnum\NameCount=0\_namecount\relax \_maybeetal \fi}
\let\editorname=\authorname
\def\prepareauedoptions#1{
  \def\maybeetal{}\csname lb@abbreviatefalse\endcsname
  \biboptionvalue{#1max}\authormax
  \biboptionvalue{#1min}\authormin
  \biboptionvalue{#1pre}\authorpre
  \biboptionvalue{#1print}\authorprint
  \isbiboption{#1etal}\iftrue\def\maybeetal{\text{bib.etal}}\fi
  \biboptionvalue{#1trim}\autrim
  \let\namecountraw=\_namecount
  \ifx\autrim\empty\def\autrim{10000}\fi
  \ifnum\autrim=0\def\autrim{10000}\fi
  \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 \nonumcitations is set).

\def\setbibmark{
  \ifx\dobibmark\undefined\def\dobibmark{}\fi
  \RetrieveFieldIn{bibmark}\_tmp
  \ifx\_tmp\empty \RetrieveFieldIn{year}\_tmp \edef\_tmp{\dobibmark, \_tmp}\fi
  \bibmark=\ea{\_tmp}\fi
}\def\bibconjunctionand{\text{bib.and}}
\def\preurl{\text{bib.available}}
\let\predoi=\preurl
\def\postedition{\text{bib.edition}}
\def\Inclause{In:~}
\def\prevolume{\text{bib.volume}}
\def\prenumber{\text{bib.number}}
\def\prepages{\text{bib.prepages}}
\def\posteditor{\ifnum0<\namecount\_relax\_Mtext{bib.editors}\_else\_Mtext{bib.editor}\_fi}
\_Mtext\{(identifier)\} expands to a phrase by outer document language (no entry language).

Non-standard field names.

Sorting.

Supporting macros.

Entry types.
2.33 Sorting and making Index

\makeindex implements sorting algorithm at TeX macro-language level. You need not any external program.

There are two passes in the sorting algorithm. The primary pass does not distinguish between a group of letters (typically non-accented and accented). If the result of comparing two strings 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 dièresis before acute before circumflex before ring. At less priority: lowercase letters must be before uppercase letters.

The |\_sortingdata|⟨iso-code⟩ implements these rules for the language ⟨iso-code⟩. The groups of characters mentioned in the |\_sortingdata|⟨iso-code⟩ (commas are ignored). The order of letters in the |\_sortingdata|⟨iso-code⟩ macro is significant for the sorting algorithm. The Czech rules (cs) are implemented here:

\def \_sortingdata{\% \( /,,.,-,@,\% \) \( a\dot{a}\dot{a}\%, \) \( bB\%, \) \( cC\%, \) \( d\dot{d}\dot{d}\%, \) \( \) \( .\) \( \) \( .\) \( \) \( .\)}
Characters ignored by the sorting algorithm are declared in \_ignoredchars\{iso-code\}. The compound characters (two or more characters interpreted as one character in the sorting algorithm) are mapped to single invisible characters in \_compoundchars\{iso-code\}. Czech rules declare ch or Ch or CH as a single letter sorted between H and I. See \_sortingdatacs above where these declared characters are used.

The characters declared in \_ignoredchars are ignored in the first pass without additional condition. All characters are taken into account in second pass: ASCII characters with code < 65 are sorted first if they are not mentioned in the \_sortingdata\{iso-code\} macro. Others not mentioned characters have undefined behavior during sorting.

Slovak sorting rules are the same as Czech. The macro \_sortingdatacs includes Slovak letters too. Compound characters are the same. English sorting rules can be defined by \_sortingdatacs too because English alphabet is a subset of the Czech and Slovak alphabets. Only difference: \_compoundcharsen is empty in English rules.

You can declare these macros for more languages if you wish to use \makeindex with sorting rules with respect to your language. Note: if you need to map compound characters to a character, don’t use \^I or \^M because these characters have very specific category codes. And use space to separate more mappings, like in \_compoundcharscs above.

Preparing to primary pass is implemented by the \_setprimarysorting macro. It is called from \makeindex macro and all processing of sorting is in a group.
Preparing to secondary pass is implemented by the \_setsecondarysorting macro.

Strings to be sorted are prepared in \_preparesorting control sequences (to save TeX memory). The \_preparesorting \_(<string>) converts \_(<string>) to \_tmpb with respect to the data initialized in \_setprimarysorting or \_setsecondarysorting. The compound characters are converted to single characters by the \_docompound macro.

Macro \_isAleB \_(<string1>) \_(<string2>) returns the result of comparison of given two strings to \_ifAleB control sequence. Usage: \_isAleB \_(<string1>) \_(<string2>) \_ifAleB ... \_else ... \_fi The converted strings (in respect of the data prepared for first pass) must be saved as values of \_(<string1>) and \_(<string2>) macros. The reason is speed: we don’t want to convert them repeatedly in each comparison. The macro \_testAleB \_(<converted-string1>) \_ifAleB \_(<converted-string2>) \_relax \_(<string1>) \_(<string2>) does the real work. It reads the first character from both converted strings, compares them and if it is equal then calls itself recursively else gives the result.
Merge sort is very effectively implemented by \TeX{} macros. The following code is created by my son Miroslav. The \texttt{\mergesort} macro expects that all items in \texttt{iilist} are separated by a comma when it starts. It ends with sorted items in \texttt{iilist} without commas. So \texttt{\dosorting} macro must prepare commas between items.

\begin{verbatim}
\def\mergesort #1#2,#3{\% by Miroslav Olsak
  \ifx,#1\% \% prazdna-skupina, neco, (#2=neco #3=pokracovani)
    \addto{iilist}{#2,}\% dvojice skupin vyresena
    \sortreturn{\if\mergesort#3\% \% mergesort pokracovani
      \fi\fi\% zarazka, na ktere se zastavi \sortreturn
    \fi\fi
  \else \% neco, prazna-skupina, (#1#2=neco #3=,)
    \addto{iilist}{#1#2,}\% dvojice skupin vyresena
    \sortreturn{\if\mergesort#3\% \% mergesort dalsi
      \fi\fi}
  \fi\% prazdne-skupinu pripadne #3 vyjmenovat
  \isAleB #1#3\% p1<p2
  \addto{iilist}{#1}\% p1 do bufferu
  \sortreturn{\if\mergesort#2,#3\% \% mergesort neco1,p2+neco2,
    \else \% p1>p2
      \addto{iilist}{#3}\% p2 do bufferu
      \sortreturn{\if\mergesort#1#2,\% \% mergesort p1+neco1,neco2,
        \fi\fi
    \fi\% koncim
  \relax \% zarazka, na ktere se zastavi \sortreturn
}
\end{verbatim}

The \texttt{\dosorting} \texttt{\list} macro redefines \texttt{\list} as sorted \texttt{\list}. The \texttt{\list} have to include control sequences in the form \texttt{\langle c\rangle\langle string\rangle}. These control sequences will be sorted with respect to \texttt{\langle string\rangle} without change of meanings of these control sequences. Their meanings are irrelevant when sorting. The first character \texttt{\langle c\rangle} in \texttt{\langle c\rangle\langle string\rangle} should be whatever. It does not influence the sorting. \OpTeX{} uses comma at this place for sorting indexes: \texttt{\langle word1\rangle,\langle word2\rangle,\langle word3\rangle \ldots}.

The actual language (chosen for hyphenation patterns) is used for sorting data. If the \texttt{\sortinglang} macro is defined as \texttt{\langle iso-code\rangle} (for example \texttt{\def\sortinglang{de}}) then this has precedence and actual language is not used. Moreover, if you specify \texttt{\asciisortingtrue} then ASCII sorting will be processed and all language sorting data will be ignored.
The \makeindex prints the index. First, it sorts the \iilist second, it prints the sorted \iilist, each item is printed using \printindexitem.

The \_printindexitem \langle word \rangle prints one item to the index. If \_, \langle word \rangle is defined then this is used instead real \langle word \rangle (this exception is declared by \iis macro). Else \langle word \rangle is printed by \printii. Finally, \printiipages prints the value of \langle word \rangle, i.e. the list of pages.

\printii \langle word \rangle& does more intelligent work because we are working with words in the form \langle main-word \rangle/\langle sub-word \rangle/\langle sub-sub-word \rangle. The \everyii tokens register is applied before \noindent. User can declare something special here.

The \newiiletter{\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:

\def\newiiletter#1#2{%
  \bigskip \hbox{\setfontsize{at15pt}\bf\uppercase{#1}}}medskip}

\printiipages \langle pglist \rangle& gets \langle pglist \rangle in the form \langle pg \rangle:\langle type \rangle, \langle pg \rangle:\langle type \rangle, \ldots \langle pg \rangle:\langle type \rangle and it converts them to \langle pg \rangle, \langle pg \rangle, \langle from\rangle--\langle to\rangle, \langle pg \rangle etc. The same pages must be printed only once and continuous consequences of pages must be compressed to the form \langle from\rangle--\langle to\rangle. Moreover, the consequence is continuous only if all pages have the same \langle type \rangle. Empty \langle type \rangle is most common, pages with b \langle type \rangle must
be printed as bold and with \textit{} as italics. Moreover, the \textit{} mentioned here are \textit{gpageno}, but we have to print \textit{pageno}. The following macros solve these tasks.

\begin{verbatim}
\def\printiipages#1&\{\let\pgtype=\undefined \tmpnum=0 \printpages #1,:,\par}
\def\printpages#1:#2,{% state automaton for comprimng pages
  \ifx,#1,\uselastpgnum
  \else \def\tmpa{#2}%
    \ifx\pgtype\tmpa \else
      \let\pgtype=\tmpa
      \uselastpgnum \usepgcomma \pgprint#1:{#2}%
    \fi
  \fi
  \ifnum\tmpnum=#1 \returnfi \fi
  \advance\tmpnum by1
  \ifnum\tmpnum=#1 \ifx\lastpgnum\undefined \usepgdash \fi \edef\lastpgnum{\the\tmpnum:{\pgtype}}%  \fi
  \uselastpgnum \usepgcomma \pgprint#1:{#2}%
  \tmpnum=#1
  \relax
\ea\printpages \fi}
\def\returnfi #1\relax{\fi}
\def\uselastpgnum{\ifx\lastpgnum\undefined
  \else \ea\pgprint\lastpgnum \let\lastpgnum=\undefined \fi}
\def\usepgcomma{\ifnum\tmpnum>0, \fi} % comma+space between page numbers
\def\usepgdash{\hbox{--}} % dash in the <from>--<to> form
\end{verbatim}

You can re-define \texttt{\pgprint} \texttt{\textlangle{}gpageno\textrangle{\textit{iitype}}} if you need to implement more \textit{iitypes}.

\begin{verbatim}
\def\pgprint #1:#2{\ifx ,#2,\pgprintA{#1}\returnfi \fi
  \ifx b#2{\bf \pgprintA{#1}}\returnfi \fi
  \ifx i#2{\it \pgprintA{#1}}\returnfi \fi
  \ifx u#2\pgu{\pgprintA{#1}}\returnfi \fi
  \pgprintA{#1}\relax}
\def\pgprintA #1#2#3#4{% #1=,\textless{}word\textgreater {} #2=,\textless{}pageno\textgreater {} #3=,\textlangle{}iitype\textrangle{ #4=
    \ifx#1\relax \global\addto\_iilist {#1}%
    \gdef#1{#2:#4}%
  \else \global\addto#1{,#2:#4}%
  \fi
  \sxdef{\pgi:#2}{#3}%
  \relax}
\end{verbatim}

The implementation of \texttt{\textlangle{}word\textrangle{\textlangle{}iitype\textrangle{}}} follows. Note that \texttt{\textlangle{}i\textrangle{}} works in the horizontal mode in order to the \texttt{\write} whatsit is not broken from the following word. If you need to keep vertical mode, use \texttt{\iiindex{\textlangle{}word\textrangle{}}} directly.

The \texttt{\_Xindex{\textlangle{}word\textrangle{}}{\textlangle{}iitype\textrangle{}}} stores \textlangle{}word\textrangle{} to the \_iilist if there is the first occurrence of the \textlangle{}word\textrangle{}. The list of pages where \textlangle{}word\textrangle{} occurs, is the value of the macro \textlangle{}word\textrangle{}, so we print \textlangle{}pageno\textrangle{} in the index, but hyperlinks are implemented by \textlangle{}gpageno\textrangle{}. So, the macro \texttt{\_pgi:} \textlangle{}gpageno\textrangle{} is defined as \textlangle{}pageno\textrangle{}.

\begin{verbatim}
\def\_iilist { }
\def \_Xindex #1#2{\_ea\_XindexA \csname \_csf\_pgi:#1\endcsname \currpage \_#2}
\if\isempty{#1}\iffalse
  \openref{\def\-{}\wref\_Xindex{#1}{\_iitypesaved}}\_#1 \fi
\public \_index ;
\end{verbatim}

The \texttt{\_iindex{\textlangle{}word\textrangle{}}} puts one \textlangle{}word\textrangle{} to the index. It writes \texttt{\_Xindex{\textlangle{}word\textrangle{}}{\textlangle{}iitype\textrangle{}}} to the .ref file. All other variants of indexing macros expand internally to \texttt{\iindex{\textlangle{}word\textrangle{}}}.

\begin{verbatim}
\def\iindex#1{\isempty{#1}\iffalse
  \openref{\def~{}\ewref\_Xindex{{#1}{\_iitypesaved}}}\fi}
\public \iindex ;
\end{verbatim}

You can re-define \texttt{\_pgprint} \texttt{\textlangle{}gpageno\textrangle{\textlangle{}iitype\textrangle{}}} if you need to implement more \textit{iitypes}.

\begin{verbatim}
\def\pgprint #1:#2{\ifx ,#2,\pgprintA{#1}\returnfi \fi
  \ifx b#2{\bf \pgprintA{#1}}\returnfi \fi
  \ifx i#2{\it \pgprintA{#1}}\returnfi \fi
  \ifx u#2\pgu{\pgprintA{#1}}\returnfi \fi
  \pgprintA{#1}\relax}
\def\pgprintA #1#2#3#4{% #1=,\textless{}word\textgreater {} #2=,\textless{}pageno\textgreater {} #3=,\textlangle{}iitype\textrangle{ #4=
    \ifx#1\relax \global\addto\_iilist {#1}%
    \gdef#1{#2:#4}%
  \else \global\addto#1{,#2:#4}%
  \fi
  \sxdef{\pgi:#2}{#3}%
  \relax}
\end{verbatim}
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.
\_ifpgfnotenote 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 0\fnotenum\or
*\or**\or***\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)} macro is simple, \fnotemark and \fnotetext does the real work.

By default \mnote{(text)} are in right margin at odd pages and they are in left margin at even pages. The \mnote macro saves its position to .ref file as \Xmnote without parameter. We define \mn:(mnotenum) as \right or \left when the .ref file is read. The \ifnum 0\leq#2 trick returns true if ⟨pageno⟩ has a numeric type and false if it is a non-numeric type (Roman numeral, for example). We prefer to use ⟨pageno⟩, but only if it has the numeric type. We use ⟨gpageno⟩ in other cases.

User can declare \fixmnotes\left or \fixmnotes\right. It defines \mnotesfixed as \left or \right which declares the placement of all marginal notes and such declaration has a precedence.

The \mnoteD{(text)} macro sets the position of the marginal note. The outer box of marginal note has zero width and zero depth and it is appended after current line using \vadjust primitive or it is inverted to vertical mode as a box with \vskip 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.

fnotes.opm
151 \newdimen\mnoteskip
152 \public \mnoteskip;

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.

fnotes.opm
160 \long\def\mnoteA #1{\incr\mnotenum
161 \ifx\motesfixed\undefined
162 \edef\motesfixed{\csname \mn:\the\mnotenum\endcsname}%
163 \else
165 \opwarning{unknown \noexpand\mnote side. TeX me again}\openref
166 \incr\unresolvedrefs
167 \def\motesfixed{\right}\%}
168 \fi \fi
169 \hbox to0pt{\ebox\rmmote{\everymnote\noindent#1\endgraf}}%
170 \dp0=0pt \box0 \kern\mnoteskip \global\mnoteskip=0pt}
171 }
172 \def \lrmnote#1#2{\ea\ifx\motesfixed\left #1\else #2\fi}

We don’t want to process \fnote, \fnotemark, \mnote in TOC, headlines nor outlines.

fnotes.opm
184 \regmacro \mdef\fnote#1{} \mdef\fnote#1{}
185 \regmacro \mdef\fnotemark#1{} \mdef\fnotemark#1{}
186 \regmacro \mdef\mnote#1{} \mdef\mnote#1{}

2.35 Styles

OpTeX provides three styles: \report, \letter and \slides. Their behavior is documented in user part of the manual in the section 1.7.2 and \slides style (for presentations) is documented in op-slides.pdf which is an example of the presentation.

2.35.1 \report and \letter styles

styles.opm
3 \codedecl \report {Basic styles of OpTeX <2021-03-10>} % preloaded in format

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 \}.

styles.opm
16 \def\boxlines{%
17 \def\boxlinesE{\ifhmode\egroup\empty\fi}%
18 \def\ni{\boxlinesE}%
19 \bgroup \lccode`\~=`\^M \lowercase{\egroup \let=}\boxlinesE
20 \everypar{\setbox0=\lastbox }\endgraf
21 \bbox{\bgroup \catcode`\~=`M=13 \let=\ni \aftergroup\boxlinesE}%
22 }
23 \def\boxlinesC{\futurelet\next\boxlinesD}
24 \def\boxlinesD{\ifx\next\empty\else\ea\egroup\fi}
25
26 \public \boxlines;

The \report and \letter style initialization macros are defined here.
The \letter defines \address and \subject macros.
The \texttt{\textbackslash slides} macro reads macro file \texttt{slides.opm}, see the section \ref{sec:2.35.2}.\pagebreak[4]

\section*{2.35.2 \texttt{\textbackslash slides} style for presentations}

Default margins and design is declared here. The \texttt{\textbackslash ttfont} is scaled by \texttt{mag1.15} in order to balance the ex height of Helvetica (Heros) and LM fonts Typewriter. The \texttt{\begtt...\endtt} verbatim is printed by smaller text.\pagebreak[1]

The bottom margin is set to 3mm. If we use 1mm, then the baseline of \texttt{\textbackslash footline} is 2mm from the bottom page. This is the depth of the \texttt{\textbackslash Grey} rectangle used for page numbers. It is r-lapped to \texttt{\textbackslash hoffset} width because left margin = \texttt{\textbackslash hoffset} = right margin. It is 14mm for narrow pages or 16mm for wide pages.
The \texttt{\_subtit} is defined analogically like \texttt{\_tit}.

The \texttt{\_pshow\{\textit{num}\}} prints the text in invisible (transparent) font when \texttt{\layernum<\textit{\textit{num}}} The transparency is set by \texttt{\_pdfpageresources} primitive.

The main level list of items is activated here. The \texttt{\_item:X} and \texttt{\_item:x} are used and are re-defined here. If we are in a nested level of items and \texttt{\_pg;+} is used then \texttt{\egroups} macro expands to the right number of \texttt{\egroup}s to close the page correctly. The level of nested item lists is saved to the \texttt{\_ilevel} register and used when we start again the next text after \texttt{\_pg;+}. 

The default values of \texttt{\_pg;}, i.e. \texttt{\_pg;}, \texttt{\_pg;+} and \texttt{\_pg;} are very simple. They are used when \texttt{\showslides} is not specified.

The \texttt{\_endslides} is defined as \texttt{\_end} primitive (preceeded by \texttt{\_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 \texttt{\_endslides}.

We need no numbers and no table of contents when using slides. The \texttt{\_printsec} macro is redefined in order the title is centered and typeset in \texttt{\Blue}.
When \slideshow is active then each page is opened by \setbox\_slidepage=\vbox\bgroup (roughly speaking) and closed by \egroup. The \_slidelayer is incremented instead instead of \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 \texttt{(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.
\texttt{\texttt{\textbackslash slideopen}} should be used instead \texttt{\textbackslash slideshow} to deactivate it but keep the borders of groups.

When \texttt{\texttt{\textbackslash slideshow}} is active then the destinations of internal hyperlinks cannot be duplicated to more “virtual” pages because hyperlink destinations have to be unique in the whole document.

The \texttt{\texttt{\textbackslash slideshow}} creates boxes of typesetting material and copies them to more pages. So, we have to suppress creating destinations in these boxes. This is done in the \texttt{\texttt{\textbackslash slidelinks}} macro. We can move creating these destinations to the output routine. \texttt{\texttt{\textbackslash destbox}} which is redefined to do only \texttt{\addto\destboxes{\texttt{\textbackslash destbox}[\texttt{\string\label}]{}}}. All destinations saved to \texttt{\texttt{\textbackslash destboxes}} are created at the start of the next output routine in the \texttt{\texttt{\textbackslash pagedest}} macro. The output routine removes \texttt{\texttt{\textbackslash destboxes}}, so each destination is created only once.

Limitations of this solution: destinations are only at the start of the page, no at the real place where \texttt{\texttt{\textbackslash wlabel}} was used. The first “virtual” page where \texttt{\texttt{\textbackslash wlabel}} is used includes its destination. If you want to go to the final page of the partially uncovering ideas then use \texttt{\texttt{\textbackslash label[\texttt{\string\label}]}\texttt{\texttt{\textbackslash wlabel}[\texttt{\string\text}]} in the last part of the page (before \texttt{\texttt{\textbackslash pg;}}) o use \texttt{\texttt{\textbackslash pgref}} instead \texttt{\ref}.

The \texttt{\texttt{\textbackslash settinglayer}} is used in the \texttt{\texttt{\textbackslash layertext}} macro to prevent printing “Duplicate label” warning when it is expanded. It is done by special value of \texttt{\texttt{\textbackslash slideshook}} used by the \texttt{\texttt{\textbackslash label}} macro). Moreover, the warning about illegal use of \texttt{\texttt{\textbackslash bib}}, \texttt{\texttt{\textbackslash usebib}} in \texttt{\texttt{\textbackslash layers}} environment is activated.

Default \texttt{\texttt{\textbackslash layers[\texttt{\string\num}]} macro (when \texttt{\texttt{\textbackslash slideshow}} is not activated) is simple. It prints the \texttt{\texttt{\textbackslash layered-text}} with \texttt{\texttt{\textbackslash layernum=\texttt{\textbackslash num}+1}} because we need the result after last layer is processed.

We must to redefine \texttt{\texttt{\textbackslash fnotenumpages}} because the data from .ref file are less usable for implementing such a feature: the footnote should be in more layers repeatedly. But we can suppose that each page starts by \texttt{\texttt{\textbackslash pg;}} macro, so we can reset the footnote counter by this macro.
2.36 Logos

Despite plain \TeX each macro for logos ends by \texttt{\ignoreslash}. This macro ignores the next slash if it is present. You can use \texttt{\TeX/} like this for protecting the space following the logo. This is visually more comfortable. The macros \TeX, \OpTeX, \LuaTeX, \XeTeX are defined.

The \texttt{\slantcorr} macro expands to the slant-correction of the current font. It is used to shifting A if the \LaTeX logo is in italic.

The expandable versions of logos used in Outlines need the expandable \texttt{\ignslash} (instead of the \texttt{\ignoreslash}).

2.37 Multilingual support

2.37.1 Lowercase, uppercase codes

All codes in Unicode table keep information about pairs lowercase-uppercase letters or single letter. We need to read such information and set appropriate \texttt{\lccode} and \texttt{\uccode}. The \texttt{\catcode} above the code 127 is not set, i.e. the \texttt{\catcode}=12 for all codes above 127.

The file \texttt{UnicodeData.txt} is read if this file exists in your \TeX distribution. The format is specified at \url{http://www.unicode.org/L2/L1999/UnicodeData.html}. We read only \texttt{Ll} (lowercase letters), \texttt{Lu} (uppercase letters) and \texttt{Lo} (other letters) and set appropriate codes. The scanner of \texttt{UnicodeData.txt} is implemented here in the group (lines 6 to 15). After the group is closed then the file \texttt{uni-lcuc.opm} is lefted by \texttt{\endinput}.

If the file \texttt{UnicodeData.txt} does not exists then internal data are used. They follow to the end of the file \texttt{uni-lcuc.opm}.
\_uterm\{Setting lccodes and uccodes for Unicode characters <2021-04-07}\} % preloaded in format.

\_isfile\{UnicodeData.txt\}\_iftrue

\begin{group}
\_sdef\{lc:Ll\}#1#2#3#4{\_global\_lccode"#2=\"2 \_global\_uccode"#2=\"0#3 }
\_sdef\{lc:Lu\}#1#2#3#4{\_global\_lccode"#2=\"0#4 \_global\_uccode"#2=\"2 }
\_sdef\{lc:Lo\}#1#2#3#4{\_global\_lccode"#2=\"2 \_global\_uccode"#2=\"2 }
\_def\pa#1;#2;#3;#4;#5;#6;#7;#8;#9;{\_ifx;#1;\_else\_ea\_pb\_fi{#1}{#3}}
\_def\pb#1#2#3;#4;#5;#6;#7;#8 {\_csname lc:#2\_endcsname\_pc{#1}{#6}{#7}\_pa}
\_def\pc#1#2#3{} % ignored if the character hasn't Ll, Lu, nor Lo type
\_everyeof={;;;;;;;;;} % end of file
\_ea\_pa\_input UnicodeData.txt
\_endgroup \_endinput \_fi % \endinput here, if UnicodeData.txt was loaded

% 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 %\endinput here, if UnicodeData.txt was loaded

\_langlist {Initialization of hyphenation patterns <2021-03-29>} % preloaded in format

The \langle iso-code \rangle means a shortcut of language name (mostly by ISO 639-1). The following control sequences are used for language switching:

- \_lan:\langle number \rangle expands to \langle iso-code \rangle of the language. The \langle number \rangle is an internal number of languages used as a value of \_language register.
- \_ulan:\langle long-lang \rangle expands to \langle iso-code \rangle too. This is transformation from long name of language (lowercase letters) to \langle iso-code \rangle.
- \_\langle iso-code \rangle Patt (for example \_csPatt) is the language \langle number \rangle declared by \chardef.
- \_\langle iso-code \rangle lang (for example \enlang, \cslang, \sklang, \delang, \pllang) is language selector. It exists in two states
  - Initialization state: when \_\langle iso-code \rangle lang is used first then it must load the patterns into memory using Lua code. If it is done then the \_\langle iso-code \rangle lang re-defines itself to the processing state.
  - Processing state: it only sets \_language=\_\langle iso-code \rangle Patt, i.e. it selects the hyphenation patterns. It does a little more language-dependent work, as mentioned below.
- \_langspecific:\langle isocode \rangle is processed by \_\langle iso-code \rangle lang and it should include language-specific macros declared by the user or macro designer.

The USenglish patters are preloaded first:

\chardef\_enPatt=0
\def\_pattlist{\_enPatt=0}
\def\_langlist{en\{USenglish\)}
\_preplang (iso-code) \langle long-lang \rangle (hyph-file-spec) \langle number \rangle \langle pre-hyph \rangle \langle post-hyph \rangle prepares the \langle iso-code \rangle lang to its initialization state. Roughly speaking, it does:

\chardef \langle iso-code \rangle Patt = \langle number \rangle
\def \langle iso-code \rangle lan: \langle number \rangle {\langle iso-code \rangle}
\def \langle iso-code \rangle lang {\_loadpattrs \langle hyph-file-spec \rangle \langle number \rangle \langle long-lang \rangle \% loads patterns using Lua code
\gdef \langle iso-code \rangle lang {\_uselang{\langle iso-code \rangle}\_\langle iso-code \rangle Patt \langle pre-hyph \rangle \langle post-hyph \rangle}
\langle iso-code \rangle lang \% runs itself in processing state
\}
\def \langle iso-code \rangle lang {\_\langle iso-code \rangle lang} \% public version

You can see that \langle iso-code \rangle lang runs \_loadpattrs and \_uselang first (in initialization state) and it runs only \_uselang when it is called again (in processing state).

\_loadpattrs (hyph-file-spec) \langle number \rangle \langle long-lang \rangle loads hyphenation patterns and hyphenation exceptions for given language and registers them as \language=\langle number \rangle.

The \langle hyph-file-spec \rangle is a part of full file name which is read: hyph-\langle hyph-file-spec \rangle.tex. The patterns and hyphenation exceptions are saved here in UTF-8 encoding. The \langle hyph-file-spec \rangle should be a list of individual \langle hyph-file-spec \rangle’s separated by commas, see the language Serbian below for an example.

\_uselang{\langle iso-code \rangle}\_\langle iso-code \rangle Patt \langle pre-hyph \rangle \langle post-hyph \rangle sets \language, \lefthyphenmin, \righthyphenmin and runs \frenchspacing. This default language-dependent settings should be re-declared by \_langspecific: \langle iso-code \rangle which is run finally (it is \relax by default, only \_langspecific:en runs \_nonfrenchspacing).
The `\uselanguage {{long-lang}}` is defined here (for compatibility with e-plain users).

```plaintext
\def\uselanguage#1{\lowercase{\cs{\ulan:#1}lang}}
\public \uselanguage ;
```

The numbers for languages are declared as fixed constants (no auto-generated). This concept is inspired by CSplain. There are typical numbers of languages in CSplain: 5=Czech in IL2, 15=Czech in T1 and 115=Czech in Unicode. We keep these constants but we load only Unicode patterns (greater than 100), of course.

```plaintext
\preplang enus USenglishmax en-us 100 23
\preplang engb UKenglish en-gb 101 23
\preplang it Italian it 102 22
\preplang ia Interlingua ia 103 22
\preplang id Indonesian id 104 22

\preplang cs Czech cs 115 23
\preplang sk Slovak sk 116 23
\preplang de German de-1996 121 22
\preplang fr French fr 122 22
\preplang pl Polish pl 123 22
\preplang cy Welsh cy 124 23
\preplang da Danish da 125 22
\preplang es Spanish es 126 22
\preplang sl Slovenian sl 128 22
\preplang fi Finnish fi 129 22
\preplang hu Hungarian hu 130 22
\preplang tr Turkish tr 131 22
\preplang et Estonian et 132 23
\preplang eu Basque eu 133 22
\preplang ga Irish ga 134 23
\preplang nb Bokmal nb 135 22
\preplang nn Nynorsk nn 136 22
\preplang nl Dutch nl 137 22
\preplang pt Portuguese pt 138 23
\preplang ro Romanian ro 139 22
\preplang hr Croatian hr 140 22
\preplang zh Pinyin zh-latn-pinyin 141 11
\preplang is Icelandic is 142 22
\preplang hsb Uppersorbian hsb 143 22
\preplang af Afrikaans af 144 12
\preplang gl Galician gl 145 22
\preplang kmr Kurmanji kmr 146 22
\preplang tk Turkmen tk 147 22
\preplang la Latin la 148 22
\preplang lac classicLatin la-x-classic 149 22
\preplang lal liturgicalLatin la-x-liturgic 150 22
\preplang elm monoGreek el-monoton 201 11
\preplang elp Greek el-polyton 202 11
\preplang grc ancientGreek grc 203 11
\preplang ca Catalan ca 204 22
\preplang cop Coptic cop 205 11
\preplang mn Mongolian mn-cyrl 206 22
\preplang sa Sanskrit sa 207 13
\preplang ru Russian ru 208 22
\preplang uk Ukrainian uk 209 22
\preplang hy Armenian hy 210 12
\preplang as Assamese as 211 11
\preplang hi Hindi hi 212 11
\preplang kn Kannada kn 213 11
\preplang lv Latvian lv 215 22
\preplang lt Lithuanian lt 216 22
\preplang ml Malayalam ml 217 11
\preplang mr Marathi mr 218 11
\preplang or Oriya or 219 11
\preplang pa Panjabi pa 220 11
\preplang ta Tamil ta 221 11
\preplang te Telugu te 222 11
```
The \langlist includes names of all languages which are ready to load and use their hyphenation patterns. This list is printed to the terminal and to log at init\TeX state here. It can be used when processing documents too.

Maybe, you need to do more language-specific actions than just switching hyphenation patterns. For example, you need to load a specific font with a specific script used in the selected language, you can define macros for quotation marks depending on the language, etc.

The example shows how to declare such language-specific things.

\def\langset #1 #2{\sdef{_langspecific:#1}{#2}}
\langset fr {... declare French quotation marks}
\langset de {... declare German quotation marks}
\langset gr {... switch to Greek fonts family}
... etc.

Note that you need not set language-specific phrases (like \today) by this code. Another concept is used for such tasks. See the section 2.37.3 for more details.

### 2.37.3 Multilingual phrases and quotation marks

Only four words are generated by Op\TeX macros: “Chapter”, “Table”, “Figure” and “Subject”. These phrases can be generated depending on the current value of \language register, if you use \mtext{⟨phrase-id⟩}, specially \mtext{chap}, \mtext{t}, \mtext{f} or \mtext{subj}. If your macros generate more words then you can define such words by \sdef{mt:⟨phrase-id⟩:⟨lang⟩} where ⟨phrase-id⟩ is a label for the declared word and ⟨lang⟩ is a language shortcut (iso code).

Using \langw{⟨lang⟩} ⟨chapter⟩ ⟨table⟩ ⟨figure⟩ ⟨subject⟩ you can declare these words more effectively:
\_langw en Chapter Table Figure Subject

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_langw cs Kapitola Tabulka Obrázek Věc</td>
</tr>
<tr>
<td>_langw de Kapitel Tabelle Abbildung Betreff</td>
</tr>
<tr>
<td>_langw es Capítulo Tabla Figura Sujeto</td>
</tr>
<tr>
<td>_langw fr Chaptire Tableau Figure Matière</td>
</tr>
<tr>
<td>_langw it Capitolo Tabella Fig. Oggetto</td>
</tr>
<tr>
<td>_langw pl Rozdział Tabela Ilustracja Temat</td>
</tr>
</tbody>
</table>

...etc. (see languages.opm)

You can add more words as you wish. For example \today macro:

\today

Quotes should be tagged by "\text" and \text\' if \iso code quotes is declared at beginning of the document (for example \enquotes). If not, then the control sequences " and \text\' are undefined. Remember, that they are used in another meaning when the \oldaccents command is used. The macros \text\n and \text\' are not defined as \protected because we need their expansion when \outlines are created. User can declare quotes by \quoteschars\langle lqq \rangle \langle rqq \rangle \langle lq \rangle \langle rq \rangle, where \langle clqq \rangle \ldots \langle crqq \rangle are normal quotes and \langle clq \rangle \ldots \langle crq \rangle are alternative quotes. or use \altquotes to swap between the meaning of these two types of quotes. \enquotes, \csquotes, \dequotes, \frquotes etc. are defined here.

The \quoteschars\langle lqq \rangle \langle rqq \rangle \langle lq \rangle \langle rq \rangle defines \" and \text\" as \qqA in normal mode and as expadable macros in outline mode. We want to well process the common cases: \"\text\&\text\" or \"\text\{\text\}. This is the reason why the quotes parameter is read in verbatim mode and retokenized again by \scantextokens. We want to allow to quote the quotes mark itself by \text\text\{\text\}. 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 ⟨lq⟩⟨rq⟩. The \_qqA qqB ⟨lqq⟩⟨rqq⟩ 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).

Sometimes should be usable to leave the markup "such" or 'such' i.e. without the first backslash. Then you can make the characters " and ' active by the _activequotes macro and leave quotes without the first backslash. First, declare _iso-code quotes, then _altquotes (if needed) and finally _activequotes.

Bibliography references generated by _usebib uses more language-dependent phrases. They are declared here. We don’t want to save all these phrases into the format, so the trick with _endinput is used here. When _usebib is processed then the following part of the file languages.opm is read again.

Only phrases of few languages are declared here now. If you want to declare phrases of your language, please create an “issue” or a “request” at https://github.com/olsak/OpTeX or send me an email with new phrases for your language (or language you know:). I am ready to put them here. Temporarily, you can put your definitions into _bibtexhook token list.

2.38 Other macros

Miscellaneous macros are here.
\texttt{\useOpTeX} and \texttt{\useoptex} are declared as \texttt{\relax}.

The \texttt{\lastpage} and \texttt{\totalpages} get the information from the \texttt{\_currpage}. The \texttt{\_Xpage} from \texttt{.ref} file sets the \texttt{\_currpage}.

We need \texttt{\uv}, \texttt{\clqq}, \texttt{\crqq}, \texttt{\flqq}, \texttt{\frqq}, \texttt{\uslang}, \texttt{\ehyph}, \texttt{\chyph}, \texttt{\shyph}, for backward compatibility with \texttt{\csplain}. Codes are set according to Unicode because we are using Czech only in Unicode when \texttt{\LUA} is used.

The \texttt{\letfont} was used in \texttt{\csplain} instead of \texttt{\fontlet}.

Non-breaking space in Unicode.

TikZ needs these funny control sequences.

We don’t want to read \texttt{\opmac.tex} unless \texttt{\input opmac} is specified.

We allow empty lines in math formulae. It is more comfortable.

Lorem ipsum can be printed by \texttt{\lipsum[(range)]} or \texttt{\lorem[(range)]}, for example \texttt{\lipsum[3]} or \texttt{\lipsum[112-121], max=150}.

First usage of \texttt{\lipsum} reads the \texttt{\LaTeX} file \texttt{lipsum.ltd.tex} by \texttt{\_lipsumload} and prints the selected paragraph(s). Next usages of \texttt{\lipsum} prints the selected paragraph(s) from memory. This second and more usages of \texttt{\lipsum} are fully expandable. If you want to have all printings of \texttt{\lipsum} expandable, use dummy \texttt{\lipsum[0]} first.

\texttt{\lipsum} adds \texttt{\par} after each printed paragraph. If you don’t need such \texttt{\par} here, use \texttt{\lipsumtext[(number)]}. This macro prints only one selected paragraph \langle number \rangle and does not add \texttt{\par}. 

\texttt{\lipsumtext} reads the \texttt{\LaTeX} file \texttt{lipsum.ltd.tex} by \texttt{\_lipsumload} and prints the selected paragraph(s). Next usages of \texttt{\lipsum} prints the selected paragraph(s) from memory. This second and more usages of \texttt{\lipsum} are fully expandable. If you want to have all printings of \texttt{\lipsum} expandable, use dummy \texttt{\lipsum[0]} first.
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 \TeX\ kernel. `luatexbase` deals with modules, allocators, and callback management. Callback management is a nice extension and is actually used in \TeX\X. Other functions are defined more or less just to suit luaotfload’s use.

```lua
local function err(message)
    error("\error: \message\n")
end

local function registernumber(name)
    return token.create(name).index
end

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", "\attribute\name\
string(cnt))
        attributes[name] = cnt
        return cnt
    end
end

provides_module = function() end

local callback_register = callback.register
function callback.register(name, fn)
    err("direct registering of callbacks is forbidden, use 'callback.add_to_callback'")
end

local callback_functions = {}
function callback.functions(name, fn)
    callback_functions[name, fn] = fn
end

local user_callbacks = {}
Table containing default functions for callbacks, which are called if either a user created callback is defined, but doesn’t have added functions or for standard callbacks that are “extended” (see mlist_to_hlist and its pre/post filters below).

```lua
local default_functions = {}

Table that maps standard (and later user) callback names to their types.

local callback_types = {
  -- file discovery
  find_read_file = "exclusive",
  find_write_file = "exclusive",
  find_font_file = "data",
  find_output_file = "data",
  find_format_file = "data",
  find_vf_file = "data",
  find_map_file = "data",
  find_enc_file = "data",
  find_pk_file = "data",
  find_data_file = "data",
  find_opentype_file = "data",
  find_truetype_file = "data",
  find_type1_file = "data",
  find_image_file = "data",

  open_read_file = "exclusive",
  read_font_file = "exclusive",
  read_vf_file = "exclusive",
  read_map_file = "exclusive",
  read_enc_file = "exclusive",
  read_pk_file = "exclusive",
  read_data_file = "exclusive",
  read_truetype_file = "exclusive",
  read_type1_file = "exclusive",
  read_opentype_file = "exclusive",

  -- data processing
  process_input_buffer = "data",
  process_output_buffer = "data",
  process_jobname = "data",
  input_level_string = "data",

  -- node list processing
  contribute_filter = "simple",
  buildpage_filter = "simple",
  build_page_insert = "exclusive",
  pre_linebreak_filter = "list",
  linebreak_filter = "exclusive",
  append_to_vlist_filter = "exclusive",
  post_linebreak_filter = "reverselist",
  hpack_filter = "list",
  vpack_filter = "list",
  hpack_quality = "list",
  vpack_quality = "list",
  process_rule = "exclusive",
  pre_output_filter = "list",
  hyphenate = "simple",
  ligaturing = "simple",
  kerning = "simple",
  insert_local_par = "simple",
  mlist_to_hlist = "exclusive",

  -- information reporting
  pre_dump = "simple",
  start_run = "simple",
  stop_run = "simple",
  start_page_number = "simple",
  stop_page_number = "simple",
  show_error_hook = "simple",
  show_error_message = "simple",
}```
show_lua_error_hook = "simple",
start_file = "simple",
stop_file = "simple",
call_edit = "simple",
finish_synctex = "simple",
wrapup_run = "simple",

-- pdf related
finish_pdffile = "data",
finish_pdfpage = "data",
page_order_index = "data",
process_pdf_image_content = "data",

-- font related
define_font = "exclusive",
glyph_not_found = "exclusive",
glyph_info = "exclusive",

-- undocumented
glyph_stream_provider = "exclusive",
provide_charproc_data = "exclusive",
}

Return a list containing descriptions of added callback functions for specific callback.

function callback.callback_descriptions(name)
  return callback_description[name] or {}
end

local valid_callback_types = {
  exclusive = true,
  simple = true,
  data = true,
  list = true,
  reverselist = true,
}

Create a user callback that can only be called manually using call_callback. A default function is only needed by "exclusive" callbacks.

function callback.create_callback(name, cbtype, default)
  if callback_types[name] then
    err("cannot create callback '"..name..'" - it already exists")
  elseif not valid_callback_types[cbtype] then
    err("cannot create callback '"..name.."' with invalid callback type '"..cbtype.."'"")
  elseif cbtype == "exclusive" and not default then
    err("unable to create exclusive callback '"..name.."', default function is required")
  end
  callback_types[name] = cbtype
  default_functions[name] = default or nil
  user_callbacks[name] = true
end

Add a function to the list of functions executed when callback is called. For standard luatex callback a proxy function that calls our machinery is registered as the real callback function. This doesn’t happen for user callbacks, that are called manually by user using call_callback or for standard callbacks that have default functions – like mlist_to_hlist (see below).

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
      err("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

190
assert(callback_register(name, function(...))
  return callback.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
    if not user_callbacks[name] and not default_functions[name] then
      -- this is a standard callback with no added functions and no
      -- default function (i.e. not mlist_to_hlist), restore standard
      -- behaviour by unregistering.
      callback_register(name, nil)
    end
  end
  return fn, description
end

helper iterator generator for iterating over reverselist callback functions

local function reverse_ipairs(t)
  local i, n = #t + 1, 1
  return function()
    i = i - 1
    if i >= n then
      return i, t[i]
    end
  end
end

Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

function callback.call_callback(name, ...)
  local cbtype = callback_types[name]
  local functions = callback_functions[name] or {default_functions[name]}
  local function reverse_ipairs(t)
    local i, n = #t + 1, 1
    return function()
      i = i - 1
      if i >= n then
        return i, t[i]
      end
    end
  end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    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
  end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    local function reverse_ipairs(t)
      local i, n = #t + 1, 1
      return function()
        i = i - 1
        if i >= n then
          return i, t[i]
        end
      end
    end

  Call all functions added to callback. This function handles standard callbacks as well as user created callbacks. It can happen that this function is called when no functions were added to callback – like for user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for empty function list.

  function callback.call_callback(name, ...)
    local cbtype = callback_types[name]
    local functions = callback_functions[name] or {default_functions[name]}
    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.
if cbtype == nil then
    err("cannot call callback ", name, '"' - no such callback exists")
elseif cbtype == "exclusive" then
    -- only one function, atleast default function is guaranteed by
    -- create_callback
    return functions[1](...)
elseif cbtype == "simple" then
    -- call all functions one after another, no passing of data
    for _, fn in ipairs(functions) do
        fn(...)
    end
    return
elseif cbtype == "data" then
    -- pass data (first argument) from one function to other, while keeping
    -- other arguments
    local data = (...)
    for _, fn in ipairs(functions) do
        data = fn(data, select(2, ...))
    end
    return data
end

-- list and reverselist are like data, but "true" keeps data (head node)
-- unchanged and "false" ends the chain immediately
local iter
if cbtype == "list" then
    iter = ipairs
elseif cbtype == "reverselist" then
    iter = reverse_ipairs
end
local head = (...)
local new_head
local changed = false
for _, fn in iter(functions) do
    new_head = fn(head, select(2, ...))
    if new_head == false then
        return false
    elseif new_head ~= true then
        head = new_head
        changed = true
    end
end
return not changed or head
end

Create “virtual” callbacks pre/post_mlist_to_hlist_filter by setting mlist_to_hlist callback.

The default behaviour of mlist_to_hlist is kept by using a default function, but it can still be overriden
by using add_to_callback.

default_functions["mlist_to_hlist"] = node.mlist_to_hlist
callback.create_callback("pre_mlist_to_hlist_filter", "list")
callback.create_callback("post_mlist_to_hlist_filter", "reverselist")
callback_register("mlist_to_hlist", function(head, ...)}
    -- pre_mlist_to_hlist_filter
    local new_head = callback.call_callback("pre_mlist_to_hlist_filter", head, ...)
    if new_head == false then
        return nil
    elseif new_head ~= true then
        head = new_head
        changed = true
    end
end
else new_head = true then
    head = new_head
end
end

-- mlist_to_hlist means either added functions or standard luatex behavior
-- of node.mlist_to_hlist (handled by default function)
head = callback.call_callback("mlist_to_hlist", head, ...)
-- post_mlist_to_hlist_filter

192
new_head = callback.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)

Compatibility with \(\LaTeX\) through luatexbase namespace. Needed for luatotload.

\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,
    callbacktypes = {}\}

2.40 Printing documentation

The \texttt{\printdoc \{filename\}\{space\}} and \texttt{\printdoctail \{filename\}\{space\}} commands are defined after the file \texttt{doc.opm} is load by \texttt{\load [doc]}.

The \texttt{\printdoc} starts reading of given \texttt{\{filename\}} from the second line. The file is read in the listing mode. The \texttt{\printdoctail} starts reading given \texttt{\{filename\}} from the first occurrence of the \_\_encode. The file is read in normal mode (like \texttt{\input \{filename\}}).

The listing mode prints the lines as a listing of a code. This mode is finished when first \_\_doc occurs or first \_\_endcode occurs. At least two spaces must precede before such \_\_doc. On the other hand, the \_\_encode must be at the left edge of the line without spaces. If this rule is not met then the listing mode continues.

If the first line or the last line of the listing mode is empty then such lines are not printed. The maximal number of printed lines in the listing mode is \texttt{\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 \texttt{\_\_doc} then the next lines are read in the normal way, but the material between \texttt{\begtt \ldots \endtt} pair is shifted by three letters left. The reason is that the three spaces of indentation is recommended in the \texttt{\_\_doc \ldots \_\_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 \texttt{\_\_endcode} then the \texttt{\endinput} is applied, the reading of the file opened by \texttt{\printdoc} is finished.

You cannot reach the end of the file (without \_\_endcode) in the listing mode.

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. Another text is printed in black.

The main documentation point is denoted by \texttt{\`(sequence)`} in red, for example \texttt{\`\foo`}. The user documentation point is the first occurrence of \texttt{\`\`\sequence`\`}, for example \texttt{\`\`\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 exists. Finally, the \texttt{\~\`(sequence)`} (for example \texttt{\~\`\foo`}) are hyperlinks to the user documentation point.

\_\_codedecl \texttt{\printdoc \{Macros for documentation printing \<2020-04-28\>}}

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 scanner of the control sequences in the listing mode.
The lines in the listing mode have a yellow background.

\def\Yellow{\setcmykcolor{0.0 0.0 0.3 0.03}}
\def\printcodeline#1{\advance\maxlines by-1
  \ifnum\maxlines<0 \ea\endverbprinting \fi
  \ifx\printfilename\relax \ttpenalty \fi \vskip-4pt
  \noindent\rlap{\Yellow \vrule height8pt depth5pt width\hsize}\
  \printfilename
  \indent \printverblinenum \par}
\def\printfilename{\hbox to0pt{\hskip\hsize\vbox to0pt{\vss\llap{\Brown\docfile}\kern7.5pt}}\hss}
\let\printfilename=\relax
\everytt={\let\printverblinenum=\relax}
\long\def\endverbprinting#1\end#2\end{\fi\fi \global\maxlines=100000
\noindent\dots etc. (see \tt\Brown\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{}\def\printdoctail #1 {\bgroup
\everytt={}\ttline=-1 \ea\printdoctailA \input #1 \egroup}{\long\gdef\printdoctailA#1\endcode{}}
\public\printdoc \printdoctail;

You can do \verb\input\vitt{\langle filename\rangle} ({\langle from\rangle}--{\langle to\rangle}) \langle filename\rangle if you need analogical design like in listing mode.

\def\vitt#1{\def\docfile{#1}\ttline=-1
  \everytt={\ttypesize[8/10]\let\printverblinenum=\relax\
  \hspace{-20pt}\mbox{\ttfamily#1}}}
\public\vitt;

The Index entries are without the trailing backslash. We must add it when printing Index.

\addto\ignoredcharsen{\_<\}
\def\printii#1#2&{\iffalse \newiiletter{#1}{#2}\def\lastii{#1}\fi
  \gdef\currii{#1#2}\the\everyii\noindent
  \hskip-\iindent \ignorespaces \printiiA\bslash#1#2\.'/}
\def\printiipages#1&{\let\pgtype=\undefined \tmpnum=0
  {\rm\printpages #1,:,\par}}
\sdef{_tocl:1}#1#2#3{\bf\llaptoclink{#1}{#2}\hfill \pgn{#3}\tocpar\medskip}
The <something> will be print as \langle something\rangle.
If this macro is loaded by \texttt{\load} then we need to initialize catcodes using the \texttt{\_afterload} macro.

\begin{verbatim}
\_def\_afterload{\catcode`\<=13 \catcode`\>=13 }
\end{verbatim}

Main documentation points and hyperlinks to/from it. Main documentation point: `\foo`. User-level documentation point: `\^\foo`, first occurrence only. The next occurrences are only links to the main documentation point. Link to user-level documentation point: `\~\foo`. If user-level documentation point follows the main documentation point then use \texttt{\_forwardlink}\`\foo`.

\begin{verbatim}
\verbchar`
\_def\#1{\leavevmode\edef\tmp{\csstring#1}\iindex{\tmp}\
  \_ifcsname cs:\tmp\_endcsname\else \_dest[cs:\tmp]\fi
  \sxdef{cs:\tmp}{}\
  \hbox{\_ifcsname cs:^\tmp\_endcsname
    \_link[cs:^\tmp]{\Red}{\tt\csstring\tmp}\else
    {\tt\Red\csstring\tmp}\fi}{\tt\Red\csstring\tmp}\fi}
\_def\_forwardlink\`#1{{\slet{cs:^\csstring#1}{relax}\`#1}}
\_def\^\`#1{\leavevmode\edef\tmp{\csstring#1}\iindex{\tmp}\
  \hbox{\_ifcsname cs:^\tmp\_endcsname
    \_link[cs:^\tmp]{\Red}{\tt\csstring\tmp}\else
    {\tt\Red\csstring\tmp}\fi}{\tt\Red\csstring\tmp}\fi}
\_def\_cslinkA{\_ifx\_next\ea\ignoreit \else \ea\ea\ea\ea\\_cslinkA}
\_def\~\`#1{\leavevmode\edef\tmp{\csstring#1}\iindex{\tmp}\
  \hbox{\_link[cs:^\tmp]{\Blue}{\tt\string\tmp}\_futurelet\next\_cslinkA}
\_futurelet\_\_cslinkA}
\end{verbatim}
Index

_\aboveliskip 126
_\abovetitle 121, 124
_\activequotes 186
_\addcitelist 150
_\addcolor 110
_\addtocorr 102
_\address 25, 175
_\addtabitemsx 145
_\addto 28, 38, 54, 103
_\addtomodlist 76
_\adef 17, 28, 38
_\adots 85
_\advancepageno 103, 105
_\afterfi 28, 41
_\afteritcorr 102
_\afterload 52
_\allocator 39
_\allowbreak 56
_\altquotes 185–186
_\ascisortingtrue 170
_\atex 128
_\authorname 153
_\b 57
_\backgroundbox 104
_\backgroundpic 138
_\bbchar 80, 95
_\begblock 14, 27, 128
_\begitemenv 13–14, 27, 48, 127
_\begitemenv 19, 27, 48, 147
_\begoutput 103–104, 118
_\begtt 16–18, 27, 47, 104, 129, 131
_\begttri 129
_\belowskip 126
_\belovtitle 121, 124
_\bf 8–9, 64–65, 80, 95
_\bg 37
_\bi 8–9, 64–65, 80, 95
_\bib 20–21, 27, 151
_\bibA 151
_\bibB 151
_\bibgl 151
_\bibitem 149, 151, 153
_\bibnum 150
_\bibnum 115, 149
_\bibpoint 48, 158
_\bibp 149
_\bibr 21, 48, 149
_\bibskip 152
_\bibtexhook 48, 152
_\bibwarning 153, 156
_\big 84
_\Big 84
_\biggl 84
_\Biggl 84
_\bigg 84
_\Bigg 84
_\biggroup 84
_\Biggroup 84
_\bigitem 84
_\Bigitem 84
_\bigitem 84
_\Bigitem 84
_\bigitem 84
_\Bigitem 84
_\bigitem 84
_\Bigitem 84
_\bibskip 56
_\Bigskip 56
_\Black 108
_\Blue 21, 108
_\bmod 87
_\boldify 67, 102
_\boldmath 9, 79, 81, 90–91, 101
_\boldunimath 91
_\bordermatrix 88
_\bordermatrixwithdelims 88
_\boxlines 175
_\bp 28, 53
_\bp 53
_\bprinta 153, 156
_\bprintb 153, 156
_\bprintc 153, 156
_\bprintd 153, 156
_\bracedparam 52
_\break 56
_\Brown 108
_\bslash 38
_\buildrel 87
_\bye 38, 59
_\byehook 38
_\c 57
_\cal 80, 95
_\caption 10–12, 27, 125
_\captionsep 125
_\cases 88
_\catgexclude 78
_\catgmaths 78
_\catgonly 78
_\catsample 78
_\catcode 53
_\cdots 85
_\centerline 56
_\chap 10, 12, 17–18, 26, 53, 121, 123
_\chapfont 67, 121
_\chapx 122
_\checkexists 34
_\cyph 24, 187
_\circle 138–139
_\circleparams 50
_\cite 12, 20–21, 27, 149, 152
_\citeA 149
_\citeb 12, 116
_\citef 150
_\clipincircle 23, 141
_\clipinonov 23, 141
_\clipinpath 141
_\clq 187
_\cmkcolordef 110
_\cmkytorgb 109
_\cnvinfo 51
_\code 28, 33–34, 54
_\codes 16–17, 27, 47, 128
_\codedcl 28, 33–35
_\colnum 145
_\colorcrop 109
_\colorev 21–22, 28, 107–109, 111
_\colorevFin 109
_\colorstackpop 109
_\colorstackpush 109
_\colorstackset 109
_\colsep 48
_\commentchars 18, 130–132
_\commoncolored 110
_\completemax 103–104
_\compoundchars 168
_\compoundcharscs 168
_\compoundcharscsen 168
_\ cong 87
_\corrmsize 81, 92
_\cramped 90
_\crl 15, 143, 146
_\crl 15, 142, 144, 146
_\crl 15, 146
_\crl 15, 142, 146
_\crl 15, 142, 146
_\crrq 187
_\cs 28, 38
_\CS 180
_\cskip 10, 125
_\cs 149, 24, 181
_\cslang 180
_\cs 24, 185
_\csquotes 24, 185
_\ctablelist 51
_\currfamily 74
_\currpage 113, 117, 187
_\currstyle 90
_\curv 70, 75
_\currvar 8–9, 63–65, 67–68, 76
_\Cyan 21, 108
_\d 57
_\dbib 151
_\ddlinedata 144

197
<table>
<thead>
<tr>
<th>Table</th>
<th>Caption</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Table 1.1 Description</td>
</tr>
<tr>
<td>1.2</td>
<td>Table 1.2 Description</td>
</tr>
<tr>
<td>1.3</td>
<td>Table 1.3 Description</td>
</tr>
<tr>
<td>1.4</td>
<td>Table 1.4 Description</td>
</tr>
<tr>
<td>1.5</td>
<td>Table 1.5 Description</td>
</tr>
<tr>
<td>1.6</td>
<td>Table 1.6 Description</td>
</tr>
<tr>
<td>1.7</td>
<td>Table 1.7 Description</td>
</tr>
<tr>
<td>1.8</td>
<td>Table 1.8 Description</td>
</tr>
<tr>
<td>1.9</td>
<td>Table 1.9 Description</td>
</tr>
<tr>
<td>1.10</td>
<td>Table 1.10 Description</td>
</tr>
<tr>
<td>1.11</td>
<td>Table 1.11 Description</td>
</tr>
<tr>
<td>1.12</td>
<td>Table 1.12 Description</td>
</tr>
<tr>
<td>1.13</td>
<td>Table 1.13 Description</td>
</tr>
<tr>
<td>1.14</td>
<td>Table 1.14 Description</td>
</tr>
<tr>
<td>1.15</td>
<td>Table 1.15 Description</td>
</tr>
<tr>
<td>1.16</td>
<td>Table 1.16 Description</td>
</tr>
<tr>
<td>1.17</td>
<td>Table 1.17 Description</td>
</tr>
<tr>
<td>1.18</td>
<td>Table 1.18 Description</td>
</tr>
<tr>
<td>1.19</td>
<td>Table 1.19 Description</td>
</tr>
<tr>
<td>1.20</td>
<td>Table 1.20 Description</td>
</tr>
<tr>
<td>1.21</td>
<td>Table 1.21 Description</td>
</tr>
<tr>
<td>1.22</td>
<td>Table 1.22 Description</td>
</tr>
<tr>
<td>1.23</td>
<td>Table 1.23 Description</td>
</tr>
<tr>
<td>1.24</td>
<td>Table 1.24 Description</td>
</tr>
<tr>
<td>1.25</td>
<td>Table 1.25 Description</td>
</tr>
<tr>
<td>1.26</td>
<td>Table 1.26 Description</td>
</tr>
<tr>
<td>1.27</td>
<td>Table 1.27 Description</td>
</tr>
<tr>
<td>1.28</td>
<td>Table 1.28 Description</td>
</tr>
<tr>
<td>1.29</td>
<td>Table 1.29 Description</td>
</tr>
<tr>
<td>1.30</td>
<td>Table 1.30 Description</td>
</tr>
</tbody>
</table>