OpTEX

Format Based on Plain T\(\TeX\) and OPmac\(^1\)

Petr Olsák, 2020, 2021, 2022, 2023

http://petr.olsak.net/optex

OpTEX is Lua\(\TeX\) format with Plain \(\TeX\) and OPmac. Only Lua\(\TeX\) engine is supported.

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

The main goal of OpTEX is:

- OpTEX keeps the simplicity (like in Plain \(\TeX\) and OPmac macros).
- There is no old obscurities concerning various 8-bit encodings and various engines.
- OpTEX provides a powerful Fonts Selection System (for Unicode font families, of course).
- OpTEX supports hyphenations of all languages installed in your \(\TeX\) system.
- All features from OPmac macros are copied. For example sorting words in the Index\(^2\), reading .\(\text{bib}\) files directly\(^2\), syntax highlighting\(^2\), colors, graphics, hyperlinks, references).
- Macros are documented in the same place where code is.
- User namespace of control sequences is separated from the internal namespace of OpTEX and primitives (\texttt{\_\_foo} versus \texttt{\_foo}). The namespaces for macro writers are designed too.

If you need to customize your document or you need to use something very specific, then you can copy relevant parts of OpTEX macros into your macro file and do changes to these macros here. This is a significant difference from \LaTeX\ or Con\TeXt\, 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.

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

\footnote{All these features are implemented by \TeX\ macros, no external program is needed.}
## Contents

1 User documentation

1.1 Starting with Op\TeX

1.2 Page layout

   1.2.1 Setting the margins
   1.2.2 Concept of the default page
   1.2.3 Footnotes and marginal notes

1.3 Fonts

   1.3.1 Font families
   1.3.2 Font sizes
   1.3.3 Typesetting math

1.4 Typical elements of the document

   1.4.1 Chapters and sections
   1.4.2 Another numbered objects
   1.4.3 References
   1.4.4 Hyperlinks, outlines
   1.4.5 Lists
   1.4.6 Tables
   1.4.7 Verbatim

1.5 Autogenerated lists

   1.5.1 Table of contents
   1.5.2 Making the index
   1.5.3 Bib\TeX\ing

1.6 Graphics

   1.6.1 Colors, transparency
   1.6.2 Images
   1.6.3 PDF transformations
   1.6.4 Ovals, circles
   1.6.5 Putting images and texts wherever

1.7 Others

   1.7.1 Using more languages
   1.7.2 Pre-defined styles
   1.7.3 Loading other macro packages
   1.7.4 Lorem ipsum dolor sit
   1.7.5 Logos
   1.7.6 The last page
   1.7.7 Use Op\TeX

1.8 Summary

1.9 API for macro writers

1.10 Compatibility with Plain \TeX

1.11 Related documents

2 Technical documentation

2.1 The main initialization file

2.2 Basic principles of Op\TeX\ sources

   2.2.1 Concept of namespaces of control sequences
   2.2.2 Macro files syntax
   2.2.3 Name spaces for package writers
   2.2.4 Summary about rules for external macro files published for Op\TeX
   2.2.5 The implementation of the namespaces and macros for macro-files
Chapter 1
User documentation

1.1 Starting with OpTEX

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

\texttt{optex \texttt{document}}

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

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

A minimal document should be

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

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

A somewhat larger example with common settings should be:

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

Tady je zkušební textík v českém jazyce.
\bye
\end{verbatim}

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

1.2 Page layout

1.2.1 Setting the margins

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

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

example:

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

Parameters are:

- \texttt{⟨pg⟩} ... 1 or 2 specifies one-page or two-pages design.
- \texttt{⟨fmt⟩} ... paper format (a4, a4l, a5, letter, etc. or user defined).
- \texttt{⟨left⟩,⟨right⟩,⟨top⟩,⟨bot⟩} ... gives the amount of left, right, top and bottom margins.
- \texttt{⟨unit⟩} ... unit used for values \texttt{⟨left⟩,⟨right⟩,⟨top⟩,⟨bot⟩}.
Each of the parameters \langle left \rangle, \langle right \rangle, \langle top \rangle, \langle bot \rangle can be empty. If both \langle left \rangle and \langle right \rangle are nonempty then \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.

Op\TeX{} declares following paper formats: a4, a4l (landscape a4), a5, a5l, a3, a3l, b5, letter and user can declare another own format by \texttt{\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\times200 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 \hsize and \vsize and margins for final printing at a5 format. The setting on the second line centers the scaled typesetting area to the true a4 paper while breaking points for paragraphs and pages are unchanged. It may be usable for review printing. After the review is done, the second line can be commented out.

1.2.2 Concept of the default page

Op\TeX{} uses “output routine” for page design. It is very similar to the Plain \TeX{} output routine. There is \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{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{draft} uses this register and puts big text DRAFT as a watermark to each page. You can try it.

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

1.2.3 Footnotes and marginal notes

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

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

```
Text in a paragraph \footnote{First notice} ... % a "normal" footnote
\table{...}{...
\footemark1...
\footemark2...} % two footnotes in a box
\footetext{Second notice}
\footetext{Third notice}
... 
\table{...}{...
\footemark1...} % one footnote in a box
\footetext{Fourth notice}
```

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

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

1.3 Fonts

1.3.1 Font families

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

\textsuperscript{2} You can declare \texttt{notenumglobal} if you want footnotes numbered in whole document from one or \texttt{notenumpages} if you want footnotes numbered at each page from one. Default setting is \texttt{notenumchapters}
If you write `\fontfam[?]` then all font families registered in OpTEX are listed on the terminal and in the log file. If you write `\fontfam[catalog]` then a catalog of all fonts registered in OpTEX and available in your TEX system is printed. See also this catalog.

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

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

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

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

Example:

```
This is \em important text. % = This is \it important text.
\it This is \em important text. % = This is\rm \it \it text.
\bf This is \em important text. % = This is \bf \it \it text.
\bi This is \em important text. % = This is\bf \it \it text.
```

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

1.3.2 Font sizes

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

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

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

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

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

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

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

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

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

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

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

1.3.3 Typesetting math

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

OpTEX preloads a collection of 7bit Computer Modern math fonts and AMS fonts in its format for math typesetting. You can use them in any size and in the \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[⟨font-file⟩]} or \texttt{loadmath[⟨font-name⟩]} 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}. See AMSTeX manual or Typesetting Math with OpTEX for complete list of math symbols.

The following math alphabets are available:

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

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

The math fonts can be scaled by \texttt{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.
1.4 Typical elements of the document

1.4.1 Chapters and sections

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

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

The chapters are automatically numbered by one number, sections by two numbers (chapter.section), and subsections by three numbers. If there are no chapters then sections have only one number and subsections two. The implicit design of the titles of chapter etc. is implemented in the macros \_printchap, \_printsec and \_printsecc. A designer can simply change these macros if he/she needs another behavior.

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 \relax if you want to have corresponding lines in the source and the output.

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

1.4.2 Another numbered objects

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

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

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

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

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

This example produces:

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

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

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

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

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

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

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

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

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

---

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\} (prints the number of the referenced object) or \pgref\{label\} (prints the page number). Example:

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

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

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

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

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

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

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

1.4.4 Hyperlinks, outlines

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

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

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

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

You can use another marking of active links: by frames which are visible in the PDF viewer but invisible when the document is printed. The way to do it is to define the macros \_pgborder, \_tocborder, \_citeborder, \_refborder and \_urlborder as the triple of RGB components of the used color. Example:
By default, these macros are not defined. It means that no frames are created.

The hyperlinked footnotes can be activated by \fnotelinks\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\[\langle type\rangle:\langle label\rangle\}\{\langle text\}\]. 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 character 4.

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

If the linked text have to be different than the URL, you can use \ulink\[\langle url\rangle\}\{\langle text\}\} macro. For example: \ulink[http://petr.olsak.net/optex]\{\OpTeX/ page\} outputs to the text OpTEX page. The characters \%, \, \#, {, and } must be escaped in the \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 0 % big bullet (default)
\style - % hyphen char
\style n % numbered items 1., 2., 3., ...
\style N % numbered items 1), 2), 3), ...
\style i % numbered items (i), (ii), (iii), ...
\style I % numbered items I, II, III, IV, ...
\style a % items of type a), b), c), ...

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

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

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

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

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

1.4.6 Tables

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

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

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

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

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

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

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

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

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

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

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

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

15
The `\vspan{number}{(text)}` shifts the `(text)` down in order it looks like to be in the center of the `(number)` lines (current line is first). You can use this for creating tables like in the following example:

```
\thisstable{|8{c|}}{\crlp{3-8}
\mspan2[c]{} & \mspan3[c]{} & Singular & \mspan3[c]{} & \mspan3[c]{} & Plural & \mspan3[c]{} & \mspan3[c]{} & Neuter & Masculine & Feminine & Masculine & Feminine & Neuter & \crlp{2-8}
\vspan2[II] & \mspan6[c]{} & X & \mspan4[c]{} & X & \mspan2[c]{} & \mspan2[c]{} & X & \mspan2[c]{} & X & \mspan2[c]{} & X & \mspan4[c]{} & X \crlp{2,4-7}
\vspan2[III] & \mspan6[c]{} & X & \mspan4[c]{} & X & \mspan2[c]{} & \mspan2[c]{} & X & \mspan2[c]{} & X & \mspan2[c]{} & X & \mspan4[c]{} & X \crlp{2-8}
}\}
```

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

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

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

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

1.4.7 Verbatim

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

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

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

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

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

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

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

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

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

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

The in-line verbatim surrounded by a \verbchar doesn’t work in parameter of macros and macro definitions. (It works in titles declared by \chap, \sec etc. and in \fnote, 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 \texttt{\texttt{verbatim}} before the first opening bracket. They are processed in the local group. For example, \texttt{\verbatiminput \hspace=20cm (-) program.c}.

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

The \texttt{\verbatiminput} can be controlled by \texttt{\everytt}, \texttt{\ttindent} just like in \texttt{\begtt...\endtt}.

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

If the code is read by \texttt{\verbatiminput} and there are comment lines prefixed by two characters then you can set them by \texttt{\commentchars{⟨first⟩⟨second⟩}}. Such comments are fully interpreted by \TeX{} (i.e. not verbatim). Section 2.28.1 (page 141) says more about this feature.

1.5 Autogenerated lists

1.5.1 Table of contents

The \texttt{\maketoc} command prints the table of contents of all \texttt{\chap}, \texttt{\sec} and \texttt{\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 \texttt{\chap} or \texttt{\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:

\texttt{\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 \texttt{\regmacro{⟨case-toc⟩}{⟨case-mark⟩}{⟨case-outline⟩}}. The parameters are applied locally in given cases. The \texttt{\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:

\begin{verbatim}
% 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}}}
  \{\def\mylogo{\hbox{{\it My\LoGo}}}
  \{\def\mylogo{MyLoGo}}
\end{verbatim}

1.5.2 Making the index

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

The \texttt{\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 \texttt{\ii resistor resistor is a passive electrical component ...}

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

18
The \iid resistor is a passive electrical component ...
or:
Now we'll deal with the \iid resistor.

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

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

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

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

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

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

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

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

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

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

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

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

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

\sec Index
\begmulti 3 \makeindex \endmulti

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

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

...
The \texttt{\textbackslash is chiquadrat \{\$\chi\$-quadrat\}} creates one entry in the “dictionary of the exceptions”. The sorting is done by the \texttt{\langle equivalent\rangle} but the \texttt{\{text\}} is printed in the index entry list.

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

### 1.5.3 Bib\TeX{}Xing

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

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

The \texttt{\textbackslash rcite\[\langle labels\rangle\]} creates the same list as \texttt{\textbackslash cite\[\langle labels\rangle\]} but without the outer brackets. Example: \cite{[tbn], pg.~13} creates \cite{27}.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
\bibpart {AA}
...\cite[label1] ... \cite[label2] ... % They belong to AA bib-list
\usebib/c (simple) file.bib % generates AA bib-list numbered 1, 2, ...
% \cite prints [1], [2], ... by bib-list AA

\bibpart {BB}
...\cite[label2] ... \cite[label1] ... % They belong to BB bib-list
\bibnum=0 \usebib/c (simple) my.bib % generates BB bib-list numbered 1, 2, ...
% \cite prints [1], [2], ... by bib-list BB
```

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

## 1.6 Graphics

### 1.6.1 Colors, transparency

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

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

The color selectors work locally in groups like font selectors.

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

The basic colors \Blue, \Red, \Cyan, \Yellow etc. are defined with CMYK components using \setcmykcolor. On the other hand, you can define a color with three RGB components and \morecolors defines such RGB colors. By default, the color model isn’t converted but only stored to PDF output for each used color. Thus, there may be a mix of color models in the PDF output which is not a good idea. You can overcome this problem by declaration \onlyrgb or \onlycmy. 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\footnote{Printed output is more equal to the monitor preview especially if you are using ICC profile for your printer.} for printing.

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

```
\colordef{MyColor}{0.5 \Orange, 0.5 \Purple}
```
The linear combination is done in CMYK subtractive color space by default (RGB colors used in \colordef argument are converted first). If the resulting component is greater than 1 then it is truncated to 1. If a convex linear combination (as in the last example above) is used then it emulates color behavior on a painter’s palette. You can use \rgbcolordef instead of \colordef if you want to mix colors in the additive RGB color space. If \onlyrgb is set then \colordef works like \rgbcolordef.

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

The \coloron macro can be defined as follows:

\def\coloron#1#2#3{% 
  \setbox0=\hbox{#2#3}\
  \leavevmode \rlap{#1\strut} \vrule width\wd0\box0
}
\coloron\Yellow\Brown{Brown text on yellow background}

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

### 1.6.2 Images

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

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

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

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

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

### 1.6.3 PDF transformations

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

---

\(^6\) A powerful and free Wysiwyg editor for creating vector graphics.

\(^7\) Chose “Omit text in PDF and create LaTeX file” option.
multiplication with the current matrix so linear transformations can be composed. One linear transformation given by the \pdfsetmatrix above transforms the vector \([0, 1]\) to \([\langle a \rangle, \langle b \rangle]\) and \([1, 0]\) to \([\langle c \rangle, \langle d \rangle]\). The stack-oriented commands \pdfsave and \pdfrestore gives a possibility of storing and restoring the current transformation matrix and the position of the current point. This position has to be the same from TeX’s point of view as from the transformation point of view when \pdfrestore is processed. Due to this fact the \pdfsave\rlap{(transformed text)}\pdfrestore or something similar is recommended.

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

\begin{align*}
\pdfscale\{\text{horizontal-factor}\}\{\text{vertical-factor}\} \\
\pdfrotate\{\text{angle-in-degrees}\}
\end{align*}

These macros simply call the properly \pdfsetmatrix command.

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

First: \pdfsave \pdfrotate{30}\pdfscale{-2}{2}\rlap{text1}\pdfrestore
% text1 is scaled two times and it is reflected about vertical axis
% and next it is rotated by 30 degrees left.

second: \pdfsave \pdfscale{-2}{2}\pdfrotate{30}\rlap{text2}\pdfrestore
% text2 is rotated by 30 degrees left then it is scaled two times
% and reflected about vertical axis.

third: \pdfsave \pdfrotate{-15.3}\pdfsetmatrix{2 0 1.5 2}\rlap{text3}\%
\pdfrestore % first slanted, then rotated by 15.3 degrees right

This gives the following result. First, second, third:

\[\text{First: } \text{text1} \quad \text{second: } \text{text2} \quad \text{third: } \text{text3}\]

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

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

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

1.6.4 Ovals, circles

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

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

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

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

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

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

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

\subsection{Putting images and texts wherever}

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

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

\section{Others}

\subsection{Using more languages}

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

\begin{verbatim}
tr[Turkish] tk[Turkmen] hsb[Uppersorbian]
\end{verbatim}

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

For compatibility with $\hat{c}$\TeX, there are macros \texttt{\textbackslash{ehyph}}, \texttt{\textbackslash{chyph}}, \texttt{\textbackslash{hyph}} which are equivalent to \texttt{\textbackslash{yenlang}}, \texttt{\textbackslash{cslang}} and \texttt{\textbackslash{sklang}}.

You can switch between language patterns by \texttt{\textbackslash{lang}} commands mentioned above. Default is \texttt{\textbackslash{enlang}}.

Op\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 \texttt{\textbackslash{iso-code}lang}. Op\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 \def or \_langw commands as shown in section 2.37.2.

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

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

1.7.2 Pre-defined styles

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

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

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

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

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

The \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, scanbase]

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

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

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

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

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

1.7.4 Lorem ipsum dolor sit

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

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

1.7.5 Logos

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

\ We are using \TeX/ because it is cool. \OpTeX/ is better than \LaTeX.\n
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
\ref [labname] % link to the chapter, section, subsection, equation
\pgref [labname] % link to the page of the chapter, section, ...

\caption/t % a numbered table caption
\caption/f % a numbered caption for the picture
\eqmark % a numbered equation

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

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

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

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

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

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

\magscale{factor} % resize typesetting, line/page breaking unchanged
\margins{bottom} % margins setting
\report \letter \slides % style declaration macros
1.9 API for macro writers

All \TeX primitives and almost all \oTeX macros are accessible by two names: \verb\\foo \ (public or user namespace) and \verb\_\foo \ (private name space). For example \verb\\hbox \ and \verb\_\hbox \ means the same \TeX primitive. More about it is documented in section 2.2.1.

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

If you are writing a document or macros specific for the document, then use simply public namespace (\foo). If you are writing more general macros, then you should declare your own namespace by \verb\\namespace \ macro and you have to follow the naming discipline described in sections 2.2.1 and 2.2.3.

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

\addto \macro \{\text\} \ adds \text \ at the end of \macro \ body, \head \macro \{\text\} \ puts \text \ at the begin.

\def \macro \{\text\} \ defines \macro \ active character with meaning \{\text\}.

\after i \{\text\} \ expands \text \ to \text \ if \text \ expanded.

\basename \macro \{\text\} \ returns the name of the file currently read.

\bp \{\dimen\ expression\} \ expands \TeX \ dimension to decimal number in \bp \ without unit.

\casesof \macro \{\text\} \{\text\} \ expands \text \ to \text \ if \text \ is case.

\colodef \macro \{\text\} \{\text\} \ declares \macro \ with \text \ and \text \ in \macro.

\cs \{\begin{string}\} \ expands \{\begin{string}\} \ to \end{string} \.

\cstoch \{\sequence\} \ expands \sequence \ to \{\begin{character}\} \ if \sequence \ was \{\begin{character}\}.

\doc \{\begin{string}\} \ expands \{\begin{string}\} \ to \{\begin{string}\} \ without \pt \ without \dimen.

\foreach \macro \{\parameters\} \{\text\} \ expands \foreach \macro \{\parameters\} \{\text\} \ to \foreach \macro \{\parameters\} \{\text\} \.

\foreach \macro \{\parameters\} \{\text\} \ expands \foreach \macro \{\parameters\} \{\text\} \ to \foreach \macro \{\parameters\} \{\text\} \.

\formun \{\begin{string}\} \ expands \{\begin{string}\} \ to \{\begin{string}\} \.

\incr \{\begin{string}\} \ expands \{\begin{string}\} \ to \{\begin{string}\} \.

\ignoret \{\begin{string}\} \ expands \{\begin{string}\} \ to \{\begin{string}\} \.

\ignoreafter \macro \{\dimen\} \ expands \macro \{\dimen\} \ to \macro \{\dimen\} \.

\isempty \{\begin{string}\} \ expands \{\begin{string}\} \ to \{\begin{string}\} \.

\isokv \{\begin{string}\} \ expands \{\begin{string}\} \ to \{\begin{string}\} \.

\loop \{\begin{string}\} \ expands \{\begin{string}\} \ to \{\begin{string}\} \.

\mathstyles \{\begin{math\} \ expands \{\begin{math\} \ to \{\begin{math\} \.

\namespace \{\begin{pkg\} \ expands \{\begin{pkg\} \ to \{\begin{pkg\} \.

\newcount \macro \ expands \macro \ to \macro \.

\newif \macro \ expands \macro \ to \macro \.

\nostyleafter \macro \, \nostylefuturelet \: they ignore the following optional space.

\openinput \{\begin{filename}\} \ reads file like \begin{input} \ but with standard catcodes.

\optdef \macro \{\begin{opt\} \parameters\} \{\text\} \ expands \macro \ with \begin{opt\} \parameters\} \{\text\} \.

\opwarning \{\begin{text}\} \ prints \{\begin{text}\} \ to \term \ and \log \ file as warning.

\pos \{\begin{label}\} \ expands \{\begin{label}\} \ to \setpos \{\begin{label}\} \.

\private \{\begin{sequence}\} \ expands \{\begin{sequence}\} \ to \{\begin{sequence}\} \.

\public \{\begin{sequence}\} \ expands \{\begin{sequence}\} \ to \{\begin{sequence}\} \.

\replstring \macro \{\begin{string}\} \{\begin{string}\} \ expands \macro \{\begin{string}\} \{\begin{string}\} \.

\settable \ and \restoretable \ manipulate with stack of catcode tables.
\let \{stringA\} \{stringB\} behaves like \let \{stringA\}=\{stringB\}.
\xdef \{string\} \{parameters\} \{body\} behaves like \xdef \{string\} \{parameters\} \{body\}.
\trycs \{string\} \{\text\} expands \{string\} if it is defined else expands \{\text\}.
\useit \{one\}, \usesecond \{one\} \{two\} uses given parameter.
\xdef \{\text\} \{parameters\} \{body\} behaves like \xdef \{\text\} \{parameters\} \{body\}.
\trycs \{text\} \{\text\} expands \{text\} if it is defined else expands \{\text\}.
\useit \{one\}, \usesecond \{one\} \{two\} uses given parameter.
\wlog \{\text\} writes \{\text\} to .log file.
\wterm \{\text\} writes \{\text\} to the terminal and .log file.
\xargs \{what\} \{token\} \{token\} \ldots ; repeats \{what\} \{token\} for each \{token\}.

1.10 Compatibility with Plain TeX

All macros of Plain TeX are re-written in OpTeX. Common macros should work in the same sense as in original Plain TeX. Internal control sequences like \f@t are removed and mostly replaced by control sequences prefixed by _ (like _this). Only a basic set of old Plain TeX control sequences like \p@, \z@, \dimen@ are provided but not recommended for new macros.

All primitives and common macros have two control sequences with the same meaning: in prefixed and unprefixed form. For example \hbox is equal to _hbox. Internal macros of 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). Users should re-define unprefixed forms of control sequences without worries that something internal will be broken.

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

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

The text accents macros \", \', \", \u, \=, \^, \., \~ are undefined in OpTeX. Use real letters like á, í, ñ 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 secant.

1.11 Related documents

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

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

\footnote{9} Use \$\sec(x)$ to get sec(x).
Chapter 2
Technical documentation

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

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.

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

% Catcodes:
\catcode `\{=1 \catcode `\}=2 \catcode `\$=3 \catcode `\&=4 \catcode `\#=6 \catcode `\^=7 \catcode `\^^K=7 \catcode `\^^A=8 \catcode `\^^I=10 \catcode `\_=11 \catcode `\~=13 \catcode `\\^0=13 \catcode 127=12

% OpTeX version
\def\optexversion{1.11+ Mar 2023}
\def\fmtname{OpTeX}
\let\fmtversion=\optexversion
\
\end{verbatim}

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

\begin{verbatim}
% OpTeX version
\def\optexversion{1.11+ Mar 2023}
\def\fmtname{OpTeX}
\let\fmtversion=\optexversion
\end{verbatim}

We check if Lua\TeX{} engine is used at -ini state. And the "\ 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.
2.2 Basic principles of OpTEX sources

2.2.1 Concept of namespaces of control sequences

OpTEX sets the category code of the \texttt{\_} character to 11 (letter) and it is never changed. So, we can always construct multiletter control sequence names from letters \texttt{A–Z}, \texttt{a–z}, and \texttt{\_}. The \texttt{letter \_} works in math mode as a subscript constructor because it is set as math active character (see section 2.15).

We distinguish following namespaces for multiletter control sequences:

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

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

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

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

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

\begin{verbatim}
\public ⟨list of control sequences⟩ ;
\end{verbatim}

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

There is an exception of the above mentioned principle. Control sequences which are alternatives to math characters (\texttt{\alpha}, \texttt{\forall}, \texttt{\subset} etc.) are declared only in public name space if they are not used in any internal OpTEX macros.

\footnote{This is only singular exception from category codes given by plain \TeX.}
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 unpre-
fixed variant of a control sequence is declared already but we need the prefixed variant too.

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

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

It is very tempting to use control sequence names with \_ in order to distinguish more words in the
sequence name. If the first character isn’t \_ then such a name is outside private and package namespaces,
so they can be used for various purposes. For example \_my_control_sequence. But there is an exception:
control sequences in the form \{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 feature is activated because we want to write math formulae as in plain \TeX, for example:

\int_a^b \ldots is interpreted as \int_a^b
\max_M \ldots is interpreted as \max_M
\alpha_{ij} \ldots is interpreted as \alpha_{ij}

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

### 2.2.2 Macro files syntax

Segments of O\TeX macros or external macro packages are stored in files with .opm extension (means
O\TeX Macros). Your local macros should be in a normal *.tex file.

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

\_codedecl \sequence {⟨short title⟩ <⟨version⟩>}

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

0:1 [⟨file name] 1 ⟨short title⟩ <⟨version⟩>

to the log file and \TeX continues with reading the following macros. If the \sequence is defined, then
\_codedecl acts like \_endinput: this protects from reading the file twice. We suppose, that \sequence
is defined in the macro file.

It is possible to use the \_doc ... \_cod pair between the macro definitions. The documentation text
should be here. It is ignored when macros are read.

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

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

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

### 2.2.3 Name spaces for package writers

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

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

\_nspublic {list of sequences} ; does \_let\_foo = \_{\_pk}\_foo for each given sequence when \_namespace{\(\_pk\)} 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 \_macro{} and then set it to the public namespace by \_nspublic \_macro;

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

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

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

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

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

- Don’t use control sequences from the public namespace in the macro bodies if there is no explicit and documented reason to do this.
- Don’t declare control sequences in the public namespace if there are no explicit and documented reasons to do this.
- Use control sequences from OpTEX and primitive namespace in read-only mode, if there is not an explicit and documented reason to redefine them.
- Use \_{\_pk}\_\_name for your internal macros or \_\_name if the \_namespace{\(\_pk\)} is declared. See section 2.2.3.
- Use \_load (or better: \_load) for loading more external macros if you need them. Don’t use \_input explicitly in such cases. The reason is: the external macro file is not loaded twice if another macro or the user needs it explicitly too.
- Use \_codedecl as your first command in the macro file and \_endcode to close the text of macros.
- Use \_doc \_\_cod pairs for documenting the code pieces.
- You can write more documentation after the \_endcode command.
- The OpTEX catcodes are set when \_load your package (i.e. plain \TeX catcodes plus catcode of \_ is 11). If a catcode is changed during loading your package then it is forgot because \_load returns to catcodes used before loading package. If you want to offer a catcode changing for users then insert it to a macro which can be used after loading.

If the macro file accepts these recommendations then it should be named by \_filename\_ops where \_filename differs from file names used directly in OpTEX and from other published macros. This extension \_ops has precedence before \_text when the \_load macro is used.

The \_math\_ops is a good example of how an external macro file for OpTEX can look like.

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

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

---

```latex
\_codedecl \_public {Prefixing and code syntax <2022-11-25>} \_preloaded in format
```

```latex
\_let\_directlua = \_directlua
\_directlua {
\% \_enable all Tex primatives with _ prefix
\text.enableprimitives(\_\_, \text.extraprimitives(\text))
```

---

3 \_codedecl \_public {Prefixing and code syntax <2022-11-25>} \_preloaded in format
9 \_let\_directlua = \_directlua
10 \_directlua {
11 \% \_enable all TeX primatives with _ prefix
12 \text.enableprimitives(\_\_, \text.extraprimitives(\text))
34
```
\texttt{\ea} is useful shortcut for \texttt{\expandafter}. We recommend to use always the private form of \texttt{\ea} because there is high probability that \texttt{\ea} will be redefined by the user.

\texttt{\public \{\texttt{sequence}\} \{\texttt{sequence}\} \ldots ; \texttt{\let \{\texttt{sequence}\} = \_\{\texttt{sequence}\} for all sequences.}
\texttt{\private \{\texttt{sequence}\} \{\texttt{sequence}\} \ldots ; \texttt{\let \{_\texttt{sequence}\} = \{\texttt{sequence}\} for all sequences.}
\texttt{\newpublic\{do\}\{\texttt{sequence}\} prints warning if \{\texttt{sequence}\} is declared already. Then runs \{do\}\{\texttt{sequence}\}.}
\texttt{\checkexists\{where\}\{\texttt{sequence-string}\} prints error if the control sequence given by its name \{\texttt{sequence-string}\} is not declared. This check is used in \texttt{\public}, \texttt{\private}, \texttt{\_nspublic} and \texttt{\_nsprivate} macros in order to avoid mistakes in names when declaring new control sequences.}
\texttt{\xargs \{what\} \{sequence\} \{sequence\} \ldots ; \texttt{does \{what\}\{sequence\} for each sequences.}

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

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

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

2.3 \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.

\long\long\def \_ignoreit #1{}  
\long\long\def \_useit #1{#1}  
\long\long\def \_ignoresecond #1#2{#1}  
\long\long\def \_usesecond #1#2{#2}  
\_public \_ignoreit \_useit \_ignoresecond \_usesecond ;

\long\long\def \_bslash \{\_csstring\\\}  
\long\long\def \_nbb \_bslash\_bslash  
\long\long\def \_pcent \{\_csstring\%\}  
\_public \bslash \nbb \pcent ;
\def \{\langle text \rangle\} is equivalent to \def \{\langle text \rangle\}, where \{\langle text \rangle\} is a control sequence. You can use arbitrary parameter mask after \sdef \{\langle text \rangle\}, don’t put the (unwanted) space immediately after closing brace \}. \sdef \{\langle text \rangle\} is equivalent to \xdef \{\langle text \rangle\}, where \xdef \{\langle text \rangle\} is a control sequence. You can use arbitrary parameter mask after \sdef \{\langle text \rangle\}, don’t put the (unwanted) space immediately after closing brace \}. \sxdef \{\langle text \rangle\} is equivalent to \xdef \{\langle text \rangle\}. \slet \{\langle textA \rangle\} = \{\langle textB \rangle\} is equivalent to \let \\langle textA \rangle = \langle textB \rangle.\basic-macros.opm

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

\basic-macros.opm

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

\basic-macros.opm

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

\basic-macros.opm

\sdef \sxdef \slet ;

\basic-macros.opm

\increment \langle counter \rangle increases \langle counter \rangle by one globally. \decrement \langle counter \rangle decreases \langle counter \rangle by one globally.

\basic-macros.opm

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

\basic-macros.opm

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

\basic-macros.opm

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

\basic-macros.opm

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

\basic-macros.opm

39
2.5 Allocators for \TeX registers

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

Unlike in Plain\TeX, the mentioned allocators are not `\outer`.

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

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

Inserts are allocated from 254 to 201 using `\newinsert`.

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

\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
\chardef\_mailanguage = 16380 % In fact 16383, but we reserve next numbers for dummy patterns
\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\_famalloc=17 \_famalloc=42 % \newfam are 43, 44, 45, ...
\countdef\_languagealloc=18 \_languagealloc=0
\end{verbatim}

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

\begin{verbatim}
\def\_allocator #1#2#3{% \_allocator #1#2#3%
  \_incr{\_cs{#2alloc}}% \_incr{#2alloc}
  \_ifnum{\_cs{#2alloc}>\_cs{\_mai#2}}% \_ifnum{#2alloc>\_mai#2}
    \errmessage{No room for a new \_ea\string\csname #2\endcsname}%
  \else
    \global\chardef#3#1=\_cs{#2alloc}
  \fi
}
\end{verbatim}
The allocation macros \newcount, \newdimen, \newskip, \newmuskip, \newbox, \newtoks, \newfam, and \newlanguage are defined here.

\def\newcount #1{\_allocator #1{count}\_countdef}
\def\newdimen #1{\_allocator #1{dimen}\_dimendef}
\def\newskip #1{\_allocator #1{skip}\_skipdef}
\def\newmuskip #1{\_allocator #1{muskip}\_muskipdef}
\def\newbox #1{\_allocator #1{box}\_chardef}
\def\newtoks #1{\_allocator #1{toks}\_toksdef}
\def\newread #1{\_allocator #1{read}\_chardef}
\def\newwrite #1{\_allocator #1{write}\_chardef}
\def\newfam #1{\_allocator #1{fam}\_chardef}
\def\newlanguage #1{\_allocator #1{language}\_chardef}
\newcount\newdimen\newskip\newmuskip\newbox\newtoks\newfam\newlanguage
\newinsert\newmarks\newattribute\newcatcodetable
\newcount\_insertalloc\_insertalloc=255
\chardef\_insertmin=201
\def\newinsert #1{\decr\_insertalloc\ifnum\_insertalloc<\_insertmin\errmessage{No room for a new \string\insert}\else\global\chardef#1=\_insertalloc\wlog{\string#1=\string\insert\_the\_insertalloc}\fi}
\public\newinsert;

The \newinsert macro is defined differently than others.

\def\newmarks #1{\_allocator #1{marks}\_chardef}
\newcount\_marksalloc\_marksalloc=0 % start at 1, 0 is \mark
\chardef\_maimarks=\_maicount
\def\newattribute #1{\_allocator #1{attribute}\_attributedef}
\newcount\_catcodetablealloc\_catcodetablealloc=10
\chardef\_maicatcodetable=32767
\def\newcatcodetable #1{\_allocator #1{catcodetable}\_chardef}
\public\newmarks\newattribute\newcatcodetable;

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

\def\tmpnum\_newcount\newcount\_tmpnum
\def\tmpdim\_newdimen\newdimen\_tmpdim

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

The \_zo and \zoskip (equivalents to \z@ and \z@skip) are declared here and used in some internal macros of Op\TeX for improving speed.
2.6 If-macros, loops, is-macros

2.6.1 Classical \newif

The \newif macro implements boolean value. It works as in plain TeX. It means that after \newif\ifxxx you can use \ifxxxtrue or \ifxxxfalse 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).

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

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

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

2.6.2 Loops

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

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

The \loop macro keeps its original plain TeX meaning. It is not expandable and nested \loops are possible only in a \TeX group.
\foreach (a,1)(b,2)(c,3)\do {#1,#2} \do {#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 \do \do \foreach is equivalent to \foreach \do \do \foreach.

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

\fornum \do \do \fornumstep \do \do \fornum \do _fornum \do \do _fornumstep \do \do _fornum \do

The \foreach and \fornum macros can be nested and arbitrary combined. When they are nested then use #1 for the variable of nested level, #2#1 for the variable of second nested level etc. Example:

\foreach ABC \do \fornum 1..5 \do {letter: #1, number: #2}
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} \iffalse {\langle codeA \rangle} \else {\langle codeB \rangle} \fi

or

\issomething {\langle params \rangle} \iftrue {\langle codeA \rangle} \iffalse {\langle codeB \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 \iffalse or \iftrue is an integral part of this syntax because we need to keep skippable nested \if conditions.

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

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

\isequal {\langle textA \rangle} {\langle textB \rangle} \iffalse 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 text \rangle} \iffalse is true if macro is defined as \langle text \rangle. Category codes are ignored in this testing. The macro is expandable.
\def\isdefined #1#2{\ifcsname #1\endcsname \else \unskip \fi #2}
\public \isdefined ;

\isinlist \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 expandable.

\long\def\isinlist#1#2{%
\immediateassignment\long\def\isinlistA#1#2\end/\%
\relax \detokenize{#2}\relax \unskip \fi%
\ea \isinlistA#1\endlistsep#2\end/%
}
\public \isinlist ;

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

\newread \testin
\def\isfile #1{%
\openin \testin ={#1} \relax
\ifeof \testin \ea \unless
\else \closein \testin
\fi
}\public \isfile ;

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

\newifi \ifexistfam
\def\isfont#1#2{%\begingroup\suppressfontnotfounderror=1\font_testfont={#1} \relax
\ifx \testfont \nullfont \def \tmp {\existfamfalse \unless}
\else \def \tmp {\existfamtrue}\fi
\ea \endgroup \tmp #2%
\public \isfont ;

The 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 expandable.

\long\def\isnextchar#1#2#3{\immediateassignment\def\isnextcharA{\isnextcharB{#1}{#2}{#3}}%
\immediateassignment\futurelet \next \isnextcharA
}
\long\def\isnextcharB#1{⟨\ifx \next #1 \ea \ignoresecond \else \ea \usesecond \fi⟩}
\public \isnextchar ;

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

\casesof ⟨token⟩ {⟨token-1⟩ {⟨what to do if token=token-1⟩}} {⟨token-2⟩ {⟨what to do if token=token-2⟩}} ...
\finc {⟨what to do in other cases⟩}

The meaning of tokens are compared by \ifx primitive. The parts ⟨what to do⟩ can be finalized by a macro which can read more data from the input stream as its parameters.
\texttt{\textbackslash xcasesof} \texttt{⟨list of pairs⟩} extends the features of the macro \texttt{\textbackslash casesof}. Each pair from the \texttt{⟨list of pairs⟩} is in the format \texttt{⟨if statement⟩}\{⟨what to do⟩\}, only the last pair must have the different format: \texttt{\_finc \{⟨what to do else⟩\}}. The \texttt{⟨if statement⟩} can be arbitrary primitive \texttt{\textbackslash if*} condition (optionally prefixed by \texttt{\textbackslash unless}) and it must be closed in its expansion. It means that \texttt{\{\textbackslash ifnum\textbackslash mycount>0\}} is bad, \texttt{\{\textbackslash ifnum\textbackslash mycount>0 \}} is correct. Optional spaces between parameters are ignored. Example:

\begin{verbatim}
\message {The \texttt{\textbackslash tmpnum} has \texttt{\textbackslash xcasesof}}
\{\texttt{\textbackslash ifnum\textbackslash tmpnum=0 }\} \{\texttt{\textbackslash equal to zero}\}
\texttt{\_finc } \{\texttt{\textbackslash negative}\} \texttt{\value}
\end{verbatim}

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

\begin{verbatim}
\_long\_def \_xcasesof #1\{\_xcasesofA #1\_finc\}
\_long\_def \_xcasesofA #1#2\_finc #3\{
\_ifx #1\_finc \_ea\_ignoresecond\_else \_ea\_usesecond\_fi
\{#3\}#1#2\_ea\_ignoresecond\_else \_ea\_usesecond\_fi \{\_finc(#3)\}\{\_xcasesof\}\}
\_public \_xcasesof ;
\end{verbatim}

### 2.7 Setting parameters

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

- primitive registers used in build-in algorithms of \LaTeX{},
- registers declared and used by Op\TeX{} macros.

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

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

\begin{verbatim}
\_codedecl \normalbaselineskip {Parameter settings <2021-04-13>} % preloaded in format
\_parindent=20pt % indentation of paragraphs
\_pretolerance=100 % parameters used in paragraph breaking algorithm
\_tolerance=200
\_hbadness=1000
\_vbadness=1000
\_doublehyphenpenalty=10000
\_finalhyphenpenalty=5000
\_adjpenalty=10000
\_uchyph=1
\_defaulthyphenchar=`-\'
\_defaultskewchar=-1
\_bfuzz=0.1pt
\_vbfuzz=0.1pt
\_overfullrule=5pt
\_linepenalty=10 % penalty between lines inside the paragraph
\_hyphenpenalty=50 % when a word is bro-ken
\_exhyphenpenalty=50 % when the hyphenmark is used explicitly
\_binoppenalty=700 % between binary operators in math
\_relpenalty=500 % between relations in math
\_brokenpenalty=100 % after lines if they end by a broken word.
\_displaywidowpenalty=50 % before last line of paragraph if display math follows
\_predisplayskippenalty=10000 % above display math
\_postdisplaypenalty=0 % below display math
\_delimiterfactor=901 % parameter for scaling delimiters
\end{verbatim}
% \scriptspace=0.5pt \phantom{\mathspaceafterscript} used in \_setmathdimens, \_setunimathdimens instead
\_maxdepth=4pt
\_splitmaxdepth=\_maxdimen
\_boxmaxdepth=\_maxdimen
\_parskip=0pt plus 1pt
\_abovedisplayskip=12pt plus 3pt minus 9pt
\_abovedisplayshortskip=0pt plus 3pt
\_belowdisplayskip=12pt plus 3pt minus 9pt
\_belowdisplayshortskip=7pt plus 3pt minus 4pt
\_parfillskip=0pt plus 1fil
\_thinmuskip=3mu
\_medmuskip=4mu plus 2mu minus 4mu
\_thickmuskip=5mu plus 5mu

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

\_topskip=10pt % top edge of page-box to first baseline distance
\_splittopskip=10pt

2.7.2 Plain \TeX registers

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

% We also define special registers that function like parameters:
\_newskip\_smallskipamount \_smallskipamount=3pt plus 1pt minus 1pt
\_newskip\_medskipamount \_medskipamount=6pt plus 2pt minus 2pt
\_newskip\_bigskipamount \_bigskipamount=12pt plus 4pt minus 4pt
\_newskip\_normalbaselineskip \_normalbaselineskip=12pt
\_newskip\_normallineskip \_normallineskip=1pt
\_newskip\_normallineskiplimit \_normallineskiplimit=0pt
\_newcount\_jot \_jot=3pt
\_newcount\_interdisplaylinepenalty \_interdisplaylinepenalty=100
\_newcount\_interfootnotelinepenalty \_interfootnotelinepenalty=100

Plain \TeX macros for setting parameters. \_normalbaselines, \_frenchspacing, \_nonfrenchspacing.

\_def\_normalbaselines{\_lineskip=\_normallineskip
\_baselineskip=\_normalbaselineskip \_lineskiplimit=\_normallineskiplimit}
\_def\_frenchspacing{\_sfcode`.=1000 \_sfcode`?=1000 \_sfcode`!=1000
\_sfcode`.=1000 \_sfcode`?=1000 \_sfcode`!=1000
\_sfcode`.=3000 \_sfcode`?=3000 \_sfcode`!=3000
\_sfcode`.=2000 \_sfcode`?=1500 \_sfcode`!=1250}
\_def\_nonfrenchspacing{\_sfcode`.=1000 \_sfcode`?=1000 \_sfcode`!=1000
\_sfcode`.=3000 \_sfcode`?=3000 \_sfcode`!=3000
\_sfcode`.=2000 \_sfcode`?=1500 \_sfcode`!=1250}

2.7.3 Different settings than in plain \TeX

Default “baseline setting” is for 10 pt fonts (like in plain \TeX). But \_typosize and \_typoscale macros re-declare it if another font size is used. The \_nonfrenchspacing is not set by default because the author of Op\TeX is living in Europe. If you set \_enlang\_hyphenation patterns then \_nonfrenchspacing is set.

\_normalbaselines % baseline setting, 10 pt font size

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

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

More parameters accepted by \pdfximage primitive can be set in the \picparams tokens list. For example \picparams={pages3} selects page 3 from included PDF file.

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

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

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

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

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

The \ttindent gives default indentation of verbatim lines printed by \begtt...\endtt pair or by \verbinput.

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

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

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

\hicolors can include a list of \hicolor commands with re-declarations of default colors mentioned in the \_hicolors⟨name⟩ from hisyntax⟨name⟩.opm file. The user can give his/her preferences about colors for syntax highlighting by this tokens list.
The default item mark used between \begitems and \enditems is the bullet. The \defaultitem tokens list declares this default item mark. The \everyitem tokens list is applied in vertical mode at the start of each item. The \everylist tokens list is applied after the group is opened by \begitems. The \ilevel keeps the value of the current nesting level of the items list. The \olistskipamount is vertical skip above and below the items list if \ilevel=1. The \ilistskipamount is vertical skip above and below the items list if \ilevel>1. The \itemskipamount is vertical skip between list items, but not above the first and below the last.

The \tit macro includes \vglue\titskip above the title of the document. The \begmulti and \endmulti pair creates more columns. The parameter \colsep declares the space between columns. If \n columns are specified then we have \n−1 \colseps and \n columns in total \hsize. This gives the definite result of the width of the columns.

Each line in the Table of contents is printed in a group. The \everytocline tokens list is processed here before the internal \_tocl:\ Associate macro which starts printing the line.

The \bibtexhook tokens list is used inside the group when \usebib command is processed after style file is loaded and before printing bib-entries. You can re-define a behavior of the style file here or you can modify the more declaration for printing (fonts, baselineskip, etc.) or you can define specific macros used in your \bib file. The \biboptions is used in the iso690 bib-style for global options, see section 2.32.5. The \bibpart saves the name of bib-list if there are more bib-lists in single document, see section 2.32.1.

\everycapitonf is used before printing caption in figures and \everycapitont is used before printing caption in tables.

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

The \everymnote is used in the \mnote group before \noindent which immediately precedes marginal note text.
The \texttt{\mnotesize} is the horizontal size of the marginal notes.
The \texttt{\mnoteindent} is horizontal space between body-text and marginal note.

\begin{verbatim}
\_newtoks\_everymnote \_newdimen\_mnotesize \_mnotesize=20mm \% the width of the mnote paragraph
\_newdimen\_mnoteindent \_mnoteindent=10pt \% distance between mnote and text
\_public \everymnote \mnotesize \mnoteindent ;
\end{verbatim}

The \texttt{\table} parameters follow. The \texttt{\thistable} tokens list register should be given 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=\{}\texttt{\}}; 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 \texttt{\tabitem} used before first column, \texttt{\tabitemr} is \texttt{\tabitem} used 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{\tabskip} is \texttt{\tabskip} used before first column, \texttt{\tabskip} is \texttt{\tabskip} used after the last column.

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

\begin{verbatim}
\_newtoks\_everytable \_newtoks\_thistable
\_newtoks\_tabiteml \_newtoks\_tabitemr \_newtoks\_tabstrut
\_newdimen\_tablinespace \_newdimen\_vvkern \_newdimen\_hhkern \_newdimen\_tsize
\_newskip\_tabskipl \_newskip\_tabskipr
\_public \everytable \thistable \tabiteml \tabitemr \tabstrut \tablinespace \vvkern \hhkern \tsize \tabskipl \tabskipr ;
\end{verbatim}

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 like in Plain \TeX. The \texttt{\eqspace} is horizontal space put between equation systems if more columns in \texttt{\eqalign} are used.

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

\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}.

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

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

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

\begin{verbatim}
\headline=\texttt{\if\itfixed Text of headline, section: \firstmark \hss}\fi
\footline=\texttt{\if\rmfixed \pageno \hfill \fi \folio \hfil}
\end{verbatim}
The distance between the `\headline` and the top of the page text is controlled by the `\headlinedist` register. The distance between the bottom of page-text and `\footline` is `\footlinedist`. More precisely: baseline of headline and baseline of the first line in page-text have distance `\headlinedist+\topskip`. The baseline of the last line in page-text and the baseline of the headline have distance `\footlinedist`. Default values are inspired by plain \TeX.

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

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

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

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

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

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

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

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

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

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

\begin{verbatim}
\newtoks \ovalparams \newtoks \circleparams
\ovalparams={\roundness=2pt \fcolor=\Yellow \lcolor=\Red \lwidth=.5bp
\shadow=N \overlapsmargins=N \hhkern=Opt \vvkern=Opt }
\circleparams={\ratio=1 \fcolor=\Yellow \lcolor=\Red \lwidth=.5bp
\shadow=N \overlapsmargins=N \hhkern=3pt \vvkern=3pt}
\newdimen \roundness \roundness=5mm % used in \clippingoval macro
\public \ovalparams \circleparams \roundness;
\end{verbatim}

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

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

and the document can be processed by \TeX 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 \setctable \{catcode table\} pushes current catcode table to the stack and activates catcodes from the \{catcode table\}. The \restorectable returns to the saved catcodes from the catcode table stack.

The \opinput works inside the catcode table stack. It reads \optexcatcodes table and stores it to \_tmpcatcodes table. This table is actually used during \input (maybe catcodes are changed here).

Finally, \_restoretable pops the stacks and returns to the catcodes used before \opinput is run.

The implementation of the catcodetable stack follows.

The current catcodes are managed in the \_catcodetable0. If the \setctable is used first (or at the outer level of the stack), then the \_catcodetable0 is pushed to the stack and the current table is re-set to the given \{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 \optexcatcodes table and sets it to the main catcode table 0.

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

If the \_afterload macro is defined then it is run after \opinput. The catcode setting should be here. Note that catcode setting done in the loaded file is forgotten after the \opinput.
The declarator `\optdef\macro #1{⟨replacement text⟩}` defines the \macro with the optional parameter followed by normal parameters declared in ⟨params⟩. The optional parameter must be used as the first parameter in brackets [...]. If it isn’t used then ⟨opt default⟩ is taken into account. The ⟨replacement text⟩ can use \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 \unspaceafter ignores the following optional space at expand processor level using the negative \romannumeral trick. The \unspacefuturelet behaves like \futurelet primitive, but it ignores the following optional space and works at expand processor level.

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

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

The \skiptoeol macro reads the text to the end of the current line and ignores it.
The \replstring macro{\langle textA\rangle}{\langle textB\rangle} replaces all occurrences of \langle textA\rangle by \langle textB\rangle in the \macro body. The \macroname must be defined without parameters. The occurrences of \langle textA\rangle are not replaced if they are “hidden” in braces, for example ...{...\langle textA\rangle}.... The category codes in the \langle textA\rangle must exactly match.

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

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

\begin{verbatim}
def replAB{\text A \and A}
\replstring \tmp{A}{B}
\end{verbatim}

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

If you really need primitive \catcode then you can use \catcode. If you do \let\foo=a then it is not simple to return from \foo to the original character code of a. You can write \a but you cannot write \foo. The macro \cstochar\langle sequence\rangle solves this problem. If the sequence is equal to a character then it expands to this character (always with catcode 12). If it isn’t equal to a character then it expands to nothing. You can say \expanded\langle cstochar\foo\rangle if you want to extract the character code.

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

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

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

You can set the number of decimal digits after decimal point of the results of `\bp` and `\expr` by optional syntax `\bp{⟨digits⟩}(⟨dimen⟩)` and `\expr{⟨digits⟩}(⟨expression⟩)`. Default is `_decdigits.`

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

You can write `\setpos{⟨label⟩}` somewhere and the position of such `\setpos{⟨label⟩}` can be referenced by `\posx{⟨label⟩}`, `\posy{⟨label⟩}` and `\pospg{⟨label⟩}`. The first two macros expand to `x` and `y` position measured from left-bottom corner of the page (dimen values) and `\pospg{⟨label⟩}` expands to the (gpageno), i.e. to the page number counted from one at beginning of the document. These values are available in the second (and more) `\TeX` run, because the information is saved to `.ref` file and restored from it at the beginning of the `\TeX` job. If these values are not known then mentioned macros expand to 0sp, 0sp and 0. The following example implements `\linefrom{⟨label⟩}` and `\lineto{⟨label⟩}` macros.

The line connecting these two points is drawn (after second `\TeX` run):

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

This is a text.\linefrom[A]\par
This is second paragraph with a text.\lineto[A]
Try to reverse from-to and watch the changes.

The coordinates are saved to the `.ref` file in the format `\_Xpos{⟨label⟩}{⟨x-pos⟩}{⟨y-pos⟩}{⟨total-pg⟩}{⟨rel-pg⟩}`. The `\_Xpos` macro defines `\posi{⟨label⟩}` as `{⟨x-pos⟩}{⟨y-pos⟩}{⟨total-pg⟩}{⟨rel-pg⟩}` auxiliary macros. The implementation of `\setpos`, `\posx` and `\posy` macros are based on `\padsavepos`, `\pdflastxpos` and `\pdflastypos` `\pd\TeX` primitives. The `\pospg` simply reads the data from the `\_currpage` macro.

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

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

\ldef\doc #1 \skiptoeol

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

\def\docgen #1 {\ea \docgenA \input{#1.opm}}
\ldef\docgenA #1\endcode #3\doc {#1\wlog{\banner}\skiptoeol}
\public \docgen ;

2.9 Using key=value format in parameters

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

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

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

\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
\readkv\myframedefaults \readkv{\the\opt}%
\rulewidth=\kv{rule-width}
\hhkern=\kv{margins}\vukern=\kv{margins}\relax
\kv{frame-color}\frame{\kv{text-color}\strut #1}%
\egroup
}

We recommend using \optdef for defining macros with optional parameters written in \[]. Then the optional parameters are saved in the \opt tokens register. First: we read default parameters by \readkv\myframedefaults and secondly the actual parameters are read by \readkv{\the\opt}. The last setting wins. Third: the values can be used by the expandable \kv{\langle key \rangle} macro. The \kv{\langle key \rangle} returns \??? if such key is not declared.

You can use keys without values in the parameters list too. Then you can ask if the key is declared by \iskv{\langle key \rangle}\iftrue or the key is undeclared by \iskv{\langle key \rangle}\iffalse. For example, you write to your documentation of your code that user can set the draft option without the value. Then you can do

\optdef\myframe [] #1{...
\readkv\myframedefaults \readkv{\the\opt}%
\iskv{draft}\iftrue ...draft mode... \else ...final mode... \fi
...}

Maybe, you want to allow not only draft option but final option (which is opposite to draft) too and you want to apply the result from the last given option. Then \iskv doesn’t work because you can only check if both options are declared but you don’t know what one is given as last. But you can use \kvx{\langle key \rangle}{\langle code \rangle} to declare \langle code \rangle which is processed immediately when the \langle key \rangle is processed by \readkv. For example
The syntax of `\kvx{⟨key⟩}{⟨code⟩}` allows to use #1 inside the code. It is replaced by the actual ⟨value⟩. Example: \kvx{opt}{⟨message|opt is #1⟩}, then \readkv{opt=HELLO} prints “opt is HELLO”.

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

The default dictionary name (where key-value pairs are processed) is empty. You can use your specific dictionary by `\kvdict={⟨name⟩}`. Then \redakv, \kv, \iskv and \nokvx macros use this named dictionary of ⟨key⟩/⟨value⟩ pairs. Package options can be processed when `\kvdict={pkg:⟨pkg⟩}`.

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

The implementation of the \readkv{⟨list⟩} expands its parameter and does replace-strings in order to remove spaces around equal signs and after commas. Then \_kvscan reads the parameters list finished by ,\_fin, and saves values to \_kv:{⟨dict⟩}:⟨key⟩ macros. The \_kvx:{⟨dict⟩}:⟨key⟩ is processed (if it is defined) with parameter ⟨value⟩ after it.

The \kvx{⟨key⟩}{⟨code⟩} defines the \_kvx:{⟨dict⟩}:⟨key⟩ macro and \nokvx{⟨code⟩} defines the \_nokvx:{⟨dict⟩}:⟨key⟩ macro.

The \kv{⟨key⟩} expands the \_kv:{⟨dict⟩}:⟨key⟩ macro. If this macro isn’t defined then \kvunknown is processed. You can re-define it if you want.

The \iskv{⟨key⟩}{\iftrue (or \iffalse) is the test, if the ⟨key⟩ is defined in current ⟨dict⟩.

2.10 Plain \TeX\ macros

All macros from plain \TeX\ are rewritten here. Differences are mentioned in the documentation below.
The shortcuts `\chardef` are not defined in OpTeX. Use normal numbers instead of such obscurities.

The `\magstep` and `\magstephalf` are defined with `\space`, (no `\relax`), in order to be expandable.

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

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

59
protected_def _eject {_par_break}
protected_def _supereject {_par\penalty-20000}
protected_def _dosupereject (#ifnum _insertpenalties>0 % something is being held over
   _line{\kern-\topskip _nobreak \_vfill \_supereject \_fi}
def _removelastskip {_ifdim _lastskip=\zo _else \_vskip-\lastskip \_fi}
def _smallbreak \_par\ifdim _lastskip<\smallskipamount
   _removelastskip \penalty-50 \_smallskip \_fi}
def _medbreak \_par\ifdim _lastskip<\medskipamount
   _removelastskip \penalty-100 \_medskip \_fi}
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 #1{\_line{#1\hss}}
\def \_rightline #1{\_line{\hss#1}}
\def \_centerline #1{\_line{\hss#1\hss}}
\def \_rlap #1{\_hbox to \zo{#1\hss}}
\def \_llap #1{\_hbox to \zo{\hss#1}}
\def \underbar #1{$\_setbox0=\_hbox{#1}\_dp0=\zo \_math \_underline{\box0}$}

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.

\def \hidewidth \_hskip \_hideskip % for alignment entries that can stick out
\def \ialign{\_everycr={}\_tabskip=\zoskip \_halign} % initialized \halign
\newcount \_mscount
\def \multispan #1{\_omit \_mscount=#1\relax
   \loop \_ifnum \_mscount>1 \_spanA \_repeat}
\def \spanA {\_span \_omit \_advance \_mscount by-1 }

\public \hidewidth \ialign \multispan;

Tabbing macros are omitted because they are obsolete.

Indentation and others. \textindent, \item, \textitem, \narrower, \raggedright, \ttraggedright, \leavevmode.

\def \hang {\_hangindent \_parindent}
\def \textindent #1{\_indent \_lapp#1\_enspace}\_ignorespaces
\def \item {\_par \_hang \textindent}
\def \textitem #1{\_par \_indent \_hangindent2\_parindent \_textindent}
\def \narrower {\_advance \leftskip \_parindent
   \_advance \rightskip \_parindent}
\def \raggedright {\_rightskip=\zo \_relax
   \_spaceskip=.3333em \_xspaceskip=.5em\relax}
\def \ttraggedright {\_tt \_rightskip=\zo \_relax
   \_spaceskip=.3333em \_xspaceskip=.5em\relax}
\def \leavevmode {\_unhbox \_voidbox} % begins a paragraph, if necessary
\def \hang {\_hangindent \_parindent}
\def \textindent {\_indent \_lapp\_enspace}\_ignorespaces
\def \item {\_par \_hang \textindent}
\def \textitem #1{\_par \_indent \_hangindent2\_parindent \_textindent}
\def \narrower {\_advance \leftskip \_parindent
   \_advance \rightskip \_parindent}
\def \raggedright {\_rightskip=\zo \_relax
   \_spaceskip=.3333em \_xspaceskip=.5em\relax}
\def \ttraggedright {\_tt \_rightskip=\zo \_relax
   \_spaceskip=.3333em \_xspaceskip=.5em\relax}
\def \leavevmode {\_unhbox \_voidbox} % begins a paragraph, if necessary

\public \hang \textindent \item \textitem \narrower \raggedright \ttraggedright \leavevmode;

Few character codes are set for backward compatibility. But old obscurities (from plain TeX) 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.
\chardef\%=`\% \
\_let\% = \_pcent % more natural, can be used in lua codes.
\chardef\&=`\& 
\chardef\#=`\#
\chardef\$=`\$
\chardef\ss="FF 
\chardef\ae="E6 
\chardef\oe="F7 
\chardef\o="F8 
\chardef\AE="C6 
\chardef\OE="D7 
\chardef\O="D8 
\chardef\i="19 
\chardef\j="1A % dotless letters 
\chardef\aa="E5 
\chardef\AA="C5 
\chardef\S="9F 
\def\l{\_errmessage{\_usedirectly ł}}
\def\L{\_errmessage{\_usedirectly Ł}}
%\def\_{\_ifmmode \kern.06em \vbox{\hrule width.3em}\else _\fi} % obsolete
\def\_{\_hbox{_}}
\def\dag{\_errmessage{\_usedirectly †}}
\def\ddag{\_errmessage{\_usedirectly ‡}}
\def\copyright{\_errmessage{\_usedirectly ©}}
%\def\Orb{\_mathhexbox20D} % obsolete (part of Copyright)
%\def\P{\_mathhexbox27B} % obsolete
\def\_usedirectly #1{Load Unicoded font by \string\fontfam\space and use directly #1}
\def\_mathhexbox #1#2#3{\_leavevmode\_hbox{\_math\_mathchar"#1#2#3}}
\public \mathhexbox ;
The \_unichars macro is run in \initunifonts, Unicodes are used instead old plain \TeX settings.
\def\_unichars{% Plain \TeX character sequences with different codes in Unicode:
\chardef\ss=`ß
\chardef\ae=`æ \chardef\AE=`Æ
\chardef\oe=`œ \chardef\OE=`Œ
\chardef\o=`ø \chardef\O=`Ø
\chardef\aa=`å \chardef\AA=`Å
\chardef\l=`ł \chardef\L=`Ł
\chardef\i=`ı \chardef\j=`ȷ
\chardef\S=`§ \chardef\P=`¶
\chardef\dag`†
\chardef\ddag`‡
\chardef\copyright`©
}
Accents. The macros \_oalign, \d, \b, \c, \dots, are defined for backward compatibility.
\def\_oalign #1{\_leavevmode\_vtop{\_baselineskip=\_zo \_lineskip=.25ex
\_align##1\_crcr\_crcr}}
\def\_oalignA{\_lineskiplimit=\_zo \_oalign}
\def\_oalign {\_lineskiplimit=\_maxdimen \_oalign} % chars over each other
\def\_shifts #1{\_dimen0=\_maxdimen \_accent\_tgrave #1 \_kern\_ea \_ignorept \_the\_fontdimen1\_font
\_dimen0} % kern by \_1 times the current slant
\def\_d #1{\_oalignA{\_relax\_1\_crr\_hidewidth\_shifts(-1ex)\_\hidewidth}}
\def\_b #1{\_oalignA{\_relax\_1\_crr\_hidewidth\_shifts(-3ex)\_\hidewidth}}
\_box to.2ex{\_hbox{\_char\_macron\_vss}\_\hidewidth}
\def\_c #1{\_setbox0=\_hbox{\_char\_macron\_\hidewidth\_\hidewidth}}
\_public \_oalign \_d \_b \_c \_dots ;
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.
\_def\^##1{{\_accent\_circumflex ##1}}%
\_def\_.##1{{\_accent\_dotaccent ##1}}%
\_def\H##1{{\_accent\_hungarumlaut ##1}}%
\_def\~##1{{\_accent\_ttilde ##1}}%
\_def\"##1{{\_accent\_dieresis ##1}}%
\_def\r##1{{\_accent\_ring ##1}}%
}
\_public \oldaccents ;

% ec-lmr encoding (will be changed after \fontfam macro):
\_chardef\_tgrave=0
\_chardef\_tacute=1
\_chardef\_circumflex=2
\_chardef\_ttilde=3
\_chardef\_dieresis=4
\_chardef\_hungarumlaut=5
\_chardef\_ring=6
\_chardef\_caron=7
\_chardef\_tbreve=8
\_chardef\_macron=9
\_chardef\_dotaccent=10
\_chardef\_cedilla=11

\_def \_uniaccents {% accents with Unicode
\_chardef\_tgrave="0060
\_chardef\_tacute="00B4
\_chardef\_circumflex="005E
\_chardef\_ttilde="02DC
\_chardef\_dieresis="00A8
\_chardef\_hungarumlaut="02DD
\_chardef\_ring="02DA
\_chardef\_caron="02C7
\_chardef\_tbreve="02D8
\_chardef\_macron="00AF
\_chardef\_dotaccent="02D9
\_chardef\_cedilla="02B8
\_chardef\_ogonek="02DB
\_let \_uniaccents=\relax
}

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.

plain-macros.opm

\_def \_hrulefill \{"_leaders\_hrule\_hfill\}
\_def \_dotfill \{"_cleaders\_hbox{$_{\_math \_mkern1.5mu.\_mkern1.5mu}$}\_hfill\}
\_def \_rightarrowfill \{$_{\_math\_smash-\_mkern-7mu%\_cleaders\_hbox{$_{\_mkern-2mu\_smash-\_mkern-2mu}$}}\_hfill$_{\_mkern-7mu\mathord\rightarrow}$\}
\_def \_leftarrowfill \{_math\_mathord\_leftarrow\_mkern-7mu%\_cleaders\_hbox{$_{\_mkern-2mu\_smash-\_mkern-2mu}$}\_hfill$_{\_mkern-7mu\mathord\leftarrow}$\}
\_mathchardef \_braceld=37A \_mathchardef \_bracerd=37B
\_mathchardef \_bracle=37C \_mathchardef \_bracer=37D
\_def \_downbracefill \{$_{\_math \_setbox0=\hbox{$_{\_braceld}$}}\_hfill_{\_bracer}$\}
\_def \_upbracefill \{$_{\_math \_setbox0=\hbox{$_{\_bracle}$}}\_hfill_{\_bracer}$\}
\_public \_hrulefill \_dotfill \_rightarrowfill \_leftarrowfill \_downbracefill \_upbracefill ;

The last part of plain \TeX{} macros: \_magnification, \_bye. Note that math macros are defined in the math-macros.opm file (section 2.15).
\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 ;

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

2.11 Preloaded fonts for text mode

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

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

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

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

2.12 Using \font primitive directly

You can declare a new font switch by \font primitive:
\font \{\font switch\} = \langle font name \rangle \{ size spec \} \\
\% for example: \\
\font \tipa = tipa10 at12pt \% the font tipa10 at 10pt is loaded \\
\% usage: \\
\{\tipa TEXT\} \% the TEXT is printed in the loaded font.

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

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

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

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

\initunifonts
\font\crimson=[Crimson-Roman] at11pt \% the font Crimson-Regular.otf is loaded \\
\font\crimsonff=[Crimson-Roman]:+smcp:+onum at11pt \% The same font is re-loaded \\
\font \c = Crimson-Regular.otf is loaded \\
\font \f = file.afm

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

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

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

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

### 2.12.1 The \setfontsize macro

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

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

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

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

The \resizethefont resizes the currently selected font to the size given by previous \setfontsize. For example
The 10 pt text is here,
\setfontsize{at12pt} the 10 pt text is here unchanged...
\resizethefont and the 12 pt text is here.

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

The following example demonstrates the mag format of \setfontsize parameter. It is only a curious
example probably not used in practical typography.
\def\smaller{\setfontsize{mag.9}\resizethefont}
Text \smaller text \smaller text \smaller text.

The \resizethefont works with arbitrary current font, for example with the font loaded directly
by \font primitive. For example:
\ufont\tencrimson=[Crimson-Roman]:+onum \% font Crimson-Regular at 10 pt is loaded
\def\crimson{\tencrimson\resizethefont} \% \crimson uses the font size on demand
\crimson The 10 pt text is here.
\setfontsize{at12pt}
\crimson The 12 pt text is here.

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

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

2.12.3 The \fontlet declarator
We have another command for scaling: \fontlet which can resize arbitrary font given by its font switch.
\fontlet \langle new font switch \rangle = \langle given font switch \rangle \langle size spec \rangle
example:
\fontlet \bigfont = \_tenbf at15pt
The \langle given font switch \rangle must be declared previously by \font or \fontlet or \fontdef. The
\langle new font switch \rangle is declared as the same font at given \langle size spec \rangle. The equal sign in the syntax is
optional. You can declare \langle new font switch \rangle as the scaled current font by
\fontlet \langle new font switch \rangle = \font \langle size spec \rangle

2.12.4 Optical sizes
There are font families with more font files where almost the same font is implemented in various design
sizes: cmr5, cmr6, cmr7, cmr8, cmr9, cmr10, cmr12, cmr17 for example. This feature is called “optical
sizes”. Each design size is implemented in its individual font file and \OpTeX\ is able to choose right file if
various optical sizes and corresponding file names are declared for the font by \_regtfm or \_regoptsizes
command. The command \setfontsize sets the internal requirements for optical size if the parameter
is in the format at\langle dimen \rangle or mag\langle factor \rangle. Then the command \resizethefont or \fontlet or variant
selectors try to choose the font suitable for the required optical size. For example
2.12.5 Font rendering

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

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

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

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

2.12.6 Implementation of resizing

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

\initunifonts macro extends LuaTEX’s font capabilities, in order to be able to load Unicode fonts. Unfortunately, this part of OpTEX depends on the luaotfload package, which adapts ConTeXt’s generic font loader for plain TeX and LaTeX. luaotfload uses Lua functions from LaTeX’s luatexbase namespace, we provide our own replacements. \initunifonts sets itself to relax because we don’t want to do this work twice. \ufont is a shortcut of \initunifonts \font.

The \setfontsize \{\size spec\} saves the \size spec to the \sizespec macro. The \optsize value is calculated from the \size spec. If the \size spec is in the format scaled\factor then \optsize is set from \defaultoptsize. If the \size spec is in the mag\number format then the contents of the \sizespec macro is re-calculated to the at\dimen format using previous \optsize value.
The \texttt{\fontname} primitive returns the \texttt{(font file name)} optionally followed by \texttt{(size spec)}. The \texttt{\xfontname} macro expands to \texttt{(font file name)} without \texttt{(size spec)}. We need to remove the part \texttt{space} at \texttt{dimen} from \texttt{\fontname} output. The letters \texttt{at} have category 12.

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

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

The implementation detail: The \texttt{\reg: \texttt{(font id)}} is defined as the \texttt{(optical size data)} and all control sequences \texttt{\reg: \texttt{(fontname)}} from this data line have the same meaning because of the \texttt{\reversetfm} macro. The \texttt{\optfn} expands this data line and apply \texttt{\runoptfn}. This macro selects the right result from the data line by testing with the current \texttt{\optsize} value.
2.13 The Font Selection System

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

2.13.1 Terminology

We distinguish between

- **font switches**, they are declared by the \font primitive or by \fontlet or \fontdef macros, they select given font.
- **variant selectors**, there are four basic variant selectors \rm, \bf, \it, \bi, there is a special selector \currvar. More variant selectors can be declared by the \famvardef macro. They select the font depending on the given variant and on the font context (i.e. on current family and on more features given by font modifiers). In addition, OpTeX defines \tt as variant selector independent of chosen font family. It selects typewriter-like font.
- **font modifiers** are declared in a family (\cond, \caps) or are “built-in” (\setfontsize{⟨size spec⟩}, \setff{⟨features⟩}). They do appropriate change in the font context but do not select the font.
- **family selectors** (for example \Termes, \LMfonts), they are declared typically in the font family files. They enable to switch between font families, they do appropriate change in the font context but do not select the font.

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

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

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

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

2.13.2 Font families, selecting fonts

The \fontfam \[ ⟨Font Family⟩ \] opens the relevant font family file where the ⟨Font Family⟩ is declared. The family selector is defined here by rules described in the section 2.13.1. Font modifiers and variant selectors may be declared here. The loaded family is set as current and \rm variant selector is processed.

The available declared font modifiers and declared variant selectors are listed in the log file when the font family is load. Or you can print \fontfam[catalog] to show available font modifiers and variant selectors.

The font modifiers can be independent, like \cond and \light. They can be arbitrarily combined (in arbitrary order) and if the font family exposes all such sub-variants then the desired font is selected (after variant selector is used). On the other hand, there are font modifiers that negates the previous font modifier, for example: \cond, \extend. You can reset all modifiers to their initial value by the \resetmod command.
You can open more font families by more \fontfam commands. Then the general method to selecting the individual font is:

\{family selector\} \{font modifiers\} \{variant selector\}

For example:

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

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

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

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

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

2.13.3 Math Fonts

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

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

2.13.4 Declaring font commands

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

- Font switches can be declared by \font primitive (see section 2.12) or by \fontlet command (see section 2.12.3) or by \fontdef command (see sections 2.13.5). When the font switches are used then they select the given font independently of the current font context. They can be used in \output routine (for example) because we need to set fixed fonts in headers and footers.

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

- The \tt selector is documented in section 2.13.7.

- Font modifiers are “built-in” or declared by \moddef command. They do modifications in the font context but don’t select any font.

  - “built-in” font modifiers are \setfontsize (see section 2.12.1), \setff (see section 2.13.9), \setletterspace and \setwordspace (see section 2.13.10). They are independent of font family.

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

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

- Font macros can be defined arbitrarily by \def primitive by users. See an example in section 2.13.8.

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

You can declare \(\langle font-switch \rangle\) by the \fontdef command.

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

where \(\langle family selector \rangle\) and \(\langle font modifiers \rangle\) are optional and \(\langle variant selector \rangle\) is mandatory.

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

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

2.13.6 The \famvardef declarator

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

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

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

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

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

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

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

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

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

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

\[
\text{\fontfam}[\text{Heros}] \text{\fontfam}[\text{Termes}]
\]

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

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

The variant selectors (declared by \famvardef) or font modifiers (declared by \moddef) are (typically) control sequences in the public namespace (\mf, \caps). They are most often declared in font family files and they are loaded by \fontfam. A conflict with such names in the public namespace can be here. For example: if \mf is defined by a user and then \fontfam[\text{Roboto}] is used then \famvardef[\text{mf}]

70
is performed for Roboto family and the original meaning of \mf is lost. But Op\TeX prints warning about it. There are two cases:

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

### 2.13.7 The \tt variant selector

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

The \tt variant selector is used in Op\TeX internal macros \ttfont (verbatim texts) and \urlfont (printing URL’s).

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

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

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

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

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

### 2.13.8 Font commands defined by \def

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

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

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

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

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

### 2.13.9 Modifying font features

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

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

You can declare a new variant selector too:

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

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

If the used font does not support the given font feature then the font is reloaded without warning nor error, silently. The font feature is not activated.
The \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 {\caps\texttt{\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\{⟨size spec⟩\}} \% sets the font size
- \texttt{\setff\{⟨font feature⟩\}} \% adds the font feature
- \texttt{\setletterspace\{⟨number⟩\}} \% sets letter spacing
- \texttt{\setwordspace\{⟨scaling⟩\}} \% modifies word spacing

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

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

\texttt{\setwordspace\{⟨scaling⟩\}} scales the default interword space (defined in the font) and its stretching and shrinking parameters by given ⟨scaling⟩ factor. For example \texttt{\setwordspace\{2.5\}} multiplies interword space by 2.5. \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 luaotfload package (see documentation of loaotfload package for more information):

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

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

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

### 2.13.11 How to create the font family file

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

\begin{verbatim}
\_fdecl \{Heros\} \{Heros \{TeX Gyre Heros fonts based on Helvetica\}\}
{\caps \cond} {\rm \bf \it \bi} \{FiraMath\}
{\{texgyreheros-regular\}}
{\_def \fontnamegen\{\{texgyreheros\-_cond\-_curr\}\}:\_capsV\_fontfeatures}
\_wlog{\_detokenize{\
Modifiers:\^\_J
\caps ***** caps & small caps\^\_J
\cond ***** condensed variants\^\_J
}}
\_moddef \resetmod \{\fset\caps=\{}\_cond=\}\_fvars \regular \bold \italic \bolditalic \
\_moddef \caps \{\fset\caps=\{smcp;\_ffonum;\}
\_moddef \nocaps \{\fset\caps=\{}\_ffonum;\}
\_moddef \cond \{\fset\cond=cn\}
\_moddef \ncond \{\fset\cond=\{}
\_initfontfamily \% new font family must be initialized
\end{verbatim}

72
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:

  ```latex
  \famdecl [{\textsl{Name of family}}] {\textsl{Familyselector}} {\{\textsl{comments}\}}
  {\{\textsl{modifiers}\}} {\{\textsl{variant selectors}\}} {\{\textsl{comments about math fonts}\}}
  {\{\textsl{font-for-testing}\}} {\def\_fontnamegen{\{\textsl{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.operators` 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: \texttt{X\caps Y}, the letters XY must be printed without any space.

- Optionally, declare new variants by the `_famvardef` macro.

- Run `_initfontfamily` to start the family (it is mandatory).

- If math font should be loaded, use `_loadmath{\{math font\}}.`

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

```latex
\font{\{font-switch\}} {\{\fontnamegen\}} \sizespec
```

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

**Example 1**

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

```latex
\moddef\resetmod{\texttt{\_fvars Regular Bold Italic BoldItalic}}
```

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

```latex
\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 \( \textit{variant} \) in this family. Then the \_fontnamegen macro expands to \([xx-\text{\_currV}]\) and it expands to \([xx-\text{Italic}]\). The Font Selection System uses \font \{[xx-Italic]\}. This command loads the \text{xx-Italic}.\texttt{otf} font file.

See more advanced examples are in f-(\textit{family})\_opm files.

**Example 2**

The f-heros\_opm is listed here. Look at it. When Heros family is selected and \textbf{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 \_\(\text{word}\)_\textit{V} are preferred. They expand typically to their content. The macro \_\fsetV \(\text{word}\)=\{\textit{content}\} (terminated by a space) is equivalent to \def\_\(\text{word}\)_\textit{V}{\{\textit{content}\}} and you can use it in font modifiers. You can use the \_fsetV macro in more general form:

\_\fsetV \(\text{word}\)=\{\textit{value-a}\},\{\textit{value-b}\},...\{\textit{value-n}\}

...etc. terminated by a space

with obvious result \def\_\(\text{word}\)_\textit{V}{\{\textit{value-a}\}};\def\_\(\text{word}\)_\textit{V}{\{\textit{value-b}\}} etc.

**Example 3**

If both font modifiers \textit{caps}, \textit{cond} were applied in Heros family, then \def\_\textit{capsV}{+smcp;}\_\textit{ffonum;} and \def\_\textit{condV}{cn} were processed by these font modifiers. If a user needs the \textbf{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.\texttt{otf} is present in your \TeX system.

The \_\textbf{onlyif} macro

has the syntax \_\textbf{onlyif} \(\text{word}\)=\{\textit{value-a}\},\{\textit{value-b}\},...\{\textit{value-n}\}: \{\langle\textit{what}\rangle\}. It can be used inside \moddef as simple IF statement: the \textit{what} is processed only if \textit{word} has \{\textit{value-a}\} or \{\textit{value-b}\} ... or \{\textit{value-n}\}. See \texttt{f-roboto.\texttt{opm}} for examples of usage of many \_\textbf{onlyif}'s.

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

The \_\textbf{moddef} macro

has the syntax \_\textbf{moddef}\{\textit{modifier}\}{\langle\textit{what to do}\rangle}. It does more things than simple \_def:

- The modifier macros are defined as \_\textbf{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 \_\textbf{famvardef} macro has the same features.

The \_\(\text{Familyselector}\) is defined by the \_\textbf{famdecl} macro as:

\protected\def\_\(\text{Familyselector}\) {%
  \_\textbf{currfamily} \{\textit{Familyselector}\}%
  \_\textbf{fontnamegen} \{...\}% this is copied from 7-th parameter of \_\textbf{famdecl}
  \resetmod
  \langle\textit{run all family-dependent font modifiers used before Familyselector without warnings}\rangle

The \_\textbf{initfontfamily}

must be run after modifier’s decaration. It runs the \_\(\text{Familyselector}\) and it runs \_\textbf{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 \_\(\text{Familyselector}\) only in public namespace without \_ prefix. We assume that if a user re-defines them then he/she needs not them, so we have no problems. If the user’s definition was done before loading the font family file then it is re-defined and Op\TeX\ warns about it. See the end of section 2.13.4.

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

The name of \_\(\text{Familyselector}\) should begin with an uppercase letter.
Please, look at OpTeX font catalogue before you will create your font family file and use the same names for analogical font modifiers (like \cond, \caps, \sans, \mono etc.) and for extra variant selectors (like \lf, \li, \kf, \ki etc. used in Roboto font family).

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

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

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

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

### Additional notes

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

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

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

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

#### 2.13.12 How to write the font family file with optical sizes

You can use \_optname macro when \_fontnamegen in expanded. This macro is fully expandable and its input is \langle internal-template \rangle and its output is a part of the font file name \langle size-dependent-template \rangle with respect to given optical size.

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

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

In general:

```
\_regoptsizes \langle internal-template \rangle \langle general-output-template \rangle \langle resizing-data \rangle
```

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

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

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

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

\def\fontalias #1 #2 {\_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:

\_wlog{\_detokenize{\nModifiers:^^J\light ..... light weight, \bf,\bi=semibold^^J\noexpd .... no expanded, no condensed, designed for 10pt size (default)^^J\eexpd ..... expanded, designed for 6pt size^^J\expd ...... semi expanded, designed for 8pt size^^J\cond ...... semi condensed, designed for 12pt size^^J\ccond ..... condensed, designed for 17pt size^^J\osize ..... auto-sitches between \ccond \cond \noexpd \expd \eexpd by size^^J\caps ...... caps & small caps^^J}}

\_moddef \resetmod {\_fsetV li={},cond={},caps={} \_fvars regular bold italic bolditalic }
\_moddef \light { \_fsetV li=lt }
\_moddef \noexpd { \_fsetV cond=\ }
\_moddef \expd { \_fsetV cond=expd }
\_moddef \_cond { \_fsetV cond=semiexpd }
\_moddef \\_fsetV cond=cond }
\_moddef \\_fsetV caps=smp;\_ffonum; }
\_moddef \\_fsetV caps=\ }
\_moddef \\_fsetV caps=\ }
\_moddef \\_fsetV caps=\ }
\_moddef \\_fsetV caps=\ }
\_moddef \\_fsetV caps=\ }
\_moddef \_regoptsizes x ? expd \semexpd <9 () <11.1 semicond <15 cond <*>}

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

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

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

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

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

\begin{verbatim}
3 \% Version <2022-10-18>. Loaded in format and secondly on demand by \fontfam[catalog]
4 \_famtext {Special name for printing a catalog :}
3 \_faminfo [Catalogue] {Catalogue of all registered font families} {fonts-catalog} {}
3 \_famalias [Catalog]
3 \_famsrc {CTAN}
3 \_famtext {Computer Modern like family:}
3 \_famfrom {GUST}
3 \_faminfo [Latin Modern] {TeX Gyre fonts based on Computer Modern} {f-lmfonts}
3 \{ -,
3 \_famalias 
3 \_faminfo [Termes] {TeX Gyre Termes fonts based on Times} {f-termes}
3 \{ -,
3 \_famalias 
3 \_faminfo [Heros] {TeX Gyre Heros fonts based on Helvetica} {f-heros}
3 \{ -,
3 \_famalias
3 ...
3 etc.

The \texttt{\_faminfo} command has the syntax:

\begin{verbatim}
\_faminfo [{\langle Family Name\rangle}] {\langle comments\rangle} {\langle file-name\rangle}
   {\langle mod-plus-vars\rangle}
\end{verbatim}

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

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

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

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

### 2.13.14 Implementation of the Font Selection System

The main principle of the Font Selection System is: run one or more modifiers followed by \texttt{\fontsel}. Modifiers save data and \texttt{\fontsel} selects the font considering saved data. Each basic variant selector \texttt{\rm}, \texttt{\bf}, \texttt{\it}, \texttt{\bi}, and \texttt{\tt} runs internal variant modifier \texttt{\_fmodrm}, \texttt{\_fmodbf}, \texttt{\_fmodit}, \texttt{\_fmodbi} and \texttt{\_fmodtt}. These modifiers save their data to the \texttt{\_famv} macro which is \texttt{\rm} or \texttt{\bf} or \texttt{\it} or \texttt{\bi} or \texttt{\tt}. The \texttt{\currvar} selector is \texttt{\fontsel} by default, but variant selectors declared by \texttt{\famvardef} change it.
The \fontsel creates the ⟨font switch⟩ in the format \tenfamv and loads the font associated to the ⟨font switch⟩. The loading is done by:

a) \letfont⟨font switch⟩ = \savedswitch \sizespec
b) \font⟨font switch⟩ = \fontnamegen \sizespec

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

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

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

The \newfontloaded should be defined for micro-typographic configuration of fonts, for example. The mte.oml package uses it. See also OpTeX trick 0058.
A large part of the Font Selection System was re-implemented in Feb. 2022. We want to keep backward compatibility:

The \famdecl [[Family Name]] \Famselector \{comment\} \{modifiers\} \{variants\} \{math\} \{font for testing\} \famnamegen{data} runs \initunifonts, then checks if \Famselector is defined. If it is true, then closes the file by \endinput. Else it defines \Famselector and saves it to the internal \f:\currfamily:main.fam command. The macro \initfontfamily needs it. The \currfamily is set to the \Famselector because the following \moddef commands need to be in the right font family context. The \currfamily is set to the \Famselector by the \Famselector too, because \Famselector must set the right font family context. The font family context is given by the current \currfamily value and by the current meaning of the \famnamegen macro. The \mathfaminfo is saved for usage in the catalog.

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

\currV expands to a template declared by \fvars depending on the ⟨variant name⟩. Usable only of standard four variants. Next variants can be declared by the \famvardef macro. \fsetq ⟨key⟩=⟨value⟩,...,⟨key⟩=⟨value⟩ expands to \def \{⟨key⟩ \V{⟨value⟩} \} in the loop.
\onlyif ⟨key⟩=⟨value-a⟩,⟨value-b⟩,...,⟨value-z⟩: ⟨what⟩ runs ⟨what⟩ only if the \{⟨key⟩ \V{⟨value⟩} \} is defined as ⟨value-a⟩ or ⟨value-b⟩ or ... or ⟨value-z⟩.
\prepcommalist ab,⟨key⟩,cd,\fin, expands to ab,,cd, (auxiliary macro used in \onlyif).
\ffonum is a shortcut for oldstyle digits font features used in font family files. You can do \let \ffonum =\ignorespaces if you don’t want to set old digits together with \caps.
The \moddef \{\text{modifier} \}\{\text{data}\} simply speaking does \def\{\text{modifier} \}\{\text{data}\}, but we need to respect the family context. In fact, \protected\def\{\text{current family}:\text{modifier}\}\{\text{data}\} is performed and the \text{\textit{modifier}} is defined as \text{\textit{famdepend}}\{\text{modifier}\}\text{\textit{f:}}\text{\textit{currfamily}:\text{modifier}}. It expands to \text{\textit{f:}}\text{\textit{currfamily}:\text{modifier}} value if it is defined or it prints the warning. When the \text{\textit{currfamily}} value is changed then we can declare the same \text{\textit{modifier}} with a different meaning.

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

\addtomodlist\text{\textit{font modifier}} adds given modifier to \modlist macro. This list is used after \resetmod when a new family is selected by a family selector, see \resetfam macro. This allows reinitializing the same current modifiers in the font context after the family is changed.
The \famvardef \xxx {\{data\}} does, roughly speaking:
\def \xxx {\{\{data\}\}\ea\the\font\def\_currvar{\xxx}}

but the macro \xxx is declared as family-dependent. It is analogically as in \moddef. The \xxx is defined as \_famdepend\xxx{f:\_currfamily:\xxx} and \f\{\currfamily\}::\xxx is defined as mentioned. 
\famvardeftt behaves somewhat differently: it defines internal version \tt (it is used in \ttfont and \urlfont) and set \tttt to the same meaning.

The \fontfam \{Font Family\} does:
- Convert its parameter to lower case and without spaces, e.g. \langle fontfamily \rangle.
- If the file \f{fontfamily}.opm exists read it and finish.
- Try to load user defined \fams-local.opm.
- If the \langle fontfamily \rangle 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.
\texttt{\fontfamsub} \texttt{[(Family)] \{(byFamily)\}} declares automatic substitution of \texttt{(Family)} by \texttt{(byFamily)} which is done when \texttt{(Family)} is not installed. I.e. if there is no \texttt{f-\{(family\).opm} file or there is no regular font of the family installed. \texttt{\fams SUBSTITUTE} is internal macro used in \texttt{\fontfam} and \texttt{\famdecl} macros. It consumes the rest of the macro, runs \texttt{\nospacefuturelet} in order to do \texttt{\endinput} to the current \texttt{f-file} and runs \texttt{\fontfam} again. The table of such substitutions are saved in the macros \texttt{\fams:\{(family\)}.}

When the \texttt{\fams-ini.opm} or \texttt{\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 \texttt{\_listfamnames} macro or when printing a catalog then more information is used and printed.

\texttt{\_famtext} does nothing or prints the text on the terminal.

\texttt{\_faminfo} \texttt{[(Family Name) \{\{comments\}\} \{\{file-name\}\} \{\{mod-plus-vars\}\} \{\{mod-plus-vars\}\} \{\{mod-plus-vars\}\} \{\{mod-plus-vars\}\}} does \texttt{\def \_famf: \{\{familyname\}\} \{\{file-name\}\} \{\{mod-plus-vars\}\}} information on the terminal. The \texttt{\{mod-plus-vars\}} data are used when printing the font catalog. \texttt{\_famalias} \texttt{[(Family Alias) \{\{familyalias\}\} \{\{file-name\}\} \{\{mod-plus-vars\}\} \{\{mod-plus-vars\}\} \{\{mod-plus-vars\}\} \{\{mod-plus-vars\}\} \{\{mod-plus-vars\}\}.}

\newtoks \famfrom \newtoks \famsrc
\input \fams-ini.opm
\let \famfile = \undefined
\famfrom = {} \famsrc = {}

When the \texttt{\fontfam[catalog]} is used then the file \texttt{\fams-catalog.opm} is read. The macro \texttt{\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 \texttt{\catalogsample}, \texttt{\catalogmathsample}, \texttt{\catalogonly} and \texttt{\catalogexclude} of the catalog are declared here.
The font features are managed in the `\fontfeatures` macro. It expands to

- `\defaultfontfeatures` – used for each font,
- `\ffadded` – features added by `\setff`,
- `\ffcolor` – features added by `\setfontcolor` (this is obsolete)
- `\fletterspace` – features added by `\setletterspace`,
- `\ffwordspace` – features added by `\setwordspace`.

The macros `\ffadded`, `\ffcolor`, `\fletterspace`, `\ffwordspace` are empty by default.

```latex
\def \fontfeatures{\defaultfontfeatures \ffadded \ffcolor \fletterspace \ffwordspace}
\def \defaultfontfeatures {+tlig;}
\def \ffadded{}
\def \ffcolor{}
\def \fletterspace{}
\def \ffwordspace{}
```

The `\setff {⟨features⟩}` adds next font features to `\ffadded`. Usage `\setff{}` resets empty set of all `\ffadded` features.

```latex
\def \setff #1{\ifx^#1^ \def \ffadded{} \else \edef \ffadded{\ffadded #1;}; \fi}
\public \setff ;
```

`\setletterspace` is based on the special font features provided by `luaotfload` package.

The `\setwordspace` recalculates the `\fontdimen2,3,4` of the font using the `\setwsp` macro which is used by the `\fontselA` macro. It activates a dummy font feature `+Ws` too in order the font is reloded by the `\font` primitive (with independent `\fontdimen` registers). If the `\setwordspace` is used again to the same font then we need to reset `\fontdimen` registers first. It is done by `\sws:{fontname}` macro which keeps the original values of the `\fontdimen`.

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

```latex
\def \setfontcolor #1{\edef \tmp{\calculatefontcolor{#1}}% \ifx \tmp \empty \def \ffcolor{} \else \edef \ffcolor{color=\tmp;}; \fi}
\def \setletterspace #1{\if^#1^ \def \fletterspace{} \else \edef \fletterspace{letterspace=#1;}; \fi}
\def \setwordspace #1{\if^#1^ \def \setwsp#1{} \def \ffwordspace{} \else \def \setwsp{\setwspA#1/} \def \ffwordspace{+Ws;}; \fi}
\def \setwsp #1{}\def \setwspA #1/#2/#3/#4{\csname _sws:\fontname#4\endcsname \relax \ea \xdef \csname _sws:\fontname#4\endcsname{\foreach 234 \do{\fontdimen##1#4=\the\fontdimen##1#4}}\fontdimen2#4=#1\fontdimen2#4\fontdimen3#4=#2\fontdimen3#4\fontdimen4#4=#3\fontdimen4#4\def \setwspC #1/{\setwspB #1/#1/#1/}}\def \calculatefontcolor#1{\trycs{\fc:#1}{#1}} % you can define more smart macro ...\edef{\fc:red}{FF0000FF} \sdef{\fc:green}{00FF00FF} \sdef{\fc:blue}{0000FFFF} \edef{\fc:yellow}{FFFF00FF} \sdef{\fc:cyan}{00FFFF00} \sdef{\fc:magenta}{FF00FFFF} \edef{\fc:white}{FFFFFF00} \sdef{\fc:grey}{00000080} \sdef{\fc:lgrey}{00000025} \edef{\fc:black}{} % ... you can declare more colors...\public \setfontcolor \setletterspace \setwordspace ;
```

`\regoptsizes ⟨internal-template⟩ ⟨left-output⟩?⟨right-output⟩? ⟨resizing-data⟩` prepares data for using by the `\optname ⟨internal-template⟩` macro. The data are saved to the `\oz:{⟨internal-template⟩}` macro. When the `\optname` is expanded then the data are scanned by the macro `\optnameA ⟨left-output⟩?⟨right-output⟩? ⟨mid-output⟩ ⟨⟨size⟩⟩` in the loop.

`\optfontalias ⟨⟨template A⟩⟩{⟨⟨template B⟩⟩} is defined as `\let \oz:{⟨template A⟩}=\oz:{⟨template B⟩}.```
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 \O\TeX says: set three sizes by the macro \setmathsizes \[⟨text-size⟩/⟨script-size⟩/⟨scriptscript-size⟩\] and then load all math fonts in given sizes by \normalmath or \boldmath macros. For example

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

We have two math macros \normalmath for the normal shape of all math symbols and \boldmath for the bold shape of all math symbols. The second one can be used in bold titles, for example. These macros load all fonts from all given math font families.
The classical math family selectors \mit, \cal, \bbchar, \frak and \script are defined here. The \rm, \bf, \it, \bi and \tt does two things: they are variant selectors for text fonts and math family selectors for math fonts. The idea was adapted from plain \TeX. These macros are redefined when unimat-codes.omp is loaded, see the section 2.16.2.

The optical sizes of Computer Modern fonts, AMS, and other fonts are declared here. The family is saved as \fam ⟨number⟩.

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

The \langle number \rangle is saved by \_loadmathfamily, \_setmathfamily to the \_mfam.

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

\_mfactor sets scaling factor for given math fonts family related to text font size. It does the setting \_ptmunit=\langle factor \rangle \_ptunit where the \langle factor \rangle is defined by \sdef{\_mfactor:\langle family \rangle}{\langle factor \rangle}.

For example, you can set \sdef{\_mfactor:1}{0.95} if you found that this scaling of math family 1 gives better visual compatibility with used text fonts. If not declared then scaling factor is 1.

\_loadmathfamily A \_chardef \_mfam
\edef \optsizesave{\the optsize}%
\optsize=\_sizemtext \font \_optfn{#1} at \optsize \_textfont \_mfam=\_mF
\optsize=\_sizemscript \font \_optfn{#1} at \optsize \_scriptfont \_mfam=\_mF
\optsize=\_sizemsscript \font \_optfn{#1} at \optsize \_scriptscriptfont \_mfam=\_mF
\optsize=\optsizesave
\let #1=\_mF

\_ptunit = 1pt
\def \setmathparam#1#2#3{% PlainTeX sets these dimens for 10pt size only:
\_delimitershortfall=0.5\_fontdimen6\_textfont3
\_nulldelimiterspace=0.12\_fontdimen6\_textfont3
\_setmathparam \_Umathspaceafterscript \_scriptspacefactor
\_skewchar\_textfont1=127 \_skewchar\_scriptfont1=127
\_skewchar\_scriptscriptfont1=127
\_skewchar\_textfont2=48 \_skewchar\_scriptfont2=48
}

The \_setmathdimens macro is used in \normalmath or \boldmath macros. It makes math dimensions dependent on the font size (plain TeX sets them only for 10pt typesetting). The \_skewchar of some math families are set here too.

\_setmathparam \Umathspaceafterscript \langle factor \rangle sets \langle lutex-param \rangle (like \umathspaceafterscript) to values dependent on \_em of textfont, scriptfont, scriptscriptfont. The \langle factor \rangle is scaling factor of mentioned \_em.

\_ptunit = 1pt
\def \setmathparam#1#2#3{% PlainTeX sets these dimens for 10pt size only:
\_delimitershortfall=0.5\_fontdimen6\_textfont3
\_nulldelimiterspace=0.12\_fontdimen6\_textfont3
\_setmathparam \_Umathspaceafterscript \_scriptspacefactor
\_skewchar\_textfont1=127 \_skewchar\_scriptfont1=127
\_skewchar\_scriptscriptfont1=127
\_skewchar\_textfont2=48 \_skewchar\_scriptfont2=48
}

The \_setmathdimens macro is used in \normalmath or \boldmath macros. It makes math dimensions dependent on the font size (plain TeX sets them only for 10pt typesetting). The \_skewchar of some math families are set here too.

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

\def \setmathdimens{% PlainTeX sets these dimens for 10pt size only:
\_delimitershortfall=0.5\_fontdimen6\_textfont3
\_nulldelimiterspace=0.12\_fontdimen6\_textfont3
\_setmathparam \Umathspaceafterscript \_scriptspacefactor
\_skewchar\_textfont1=127 \_skewchar\_scriptfont1=127
\_skewchar\_scriptscriptfont1=127
\_skewchar\_textfont2=48 \_skewchar\_scriptfont2=48
}
Finally, we preload a math fonts collection in [10/7/5] sizes when the format is generated. This is done when `\suppressfontnotfounderror=1` because we need not errors when the format is generated. Maybe there are not all fonts in the \TeX{} distribution installed.

```latex
\suppressfontnotfounderror=1
\setmathsizes[10/7/5]
\ifsfontspreload\relax \else \normalmath \fi
\suppressfontnotfounderror=0
```

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`, `^`, `\int`, `_`, `b`. The control sequence `\int` isn’t 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 word \rangle_ \langle one-letter \rangle` (where `\langle word \rangle` is a sequence of letters) because such control sequences are inaccessible: preprocessor rewrites it.

The `\mathsbon` macro activates the rewriting rule `\langle word \rangle_ \langle nonletter \rangle` to `\langle word \rangle \langle letter \rangle \langle 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.

```latex
\catcode`\_ = 8 \let \sb = _
\catcode`\_ = 13 \let _ = \sb
\catcode`\_ = 11
\_private \sb ;
\_newifi\_ifmathsb \_mathsbfalse
\_def \_mathsbon {%
\_directlua{
\_global\_mathsbtrue
}\_directlua{
\_global\_mathsbfalse
}\_public \_mathsboff \_mathsb ;
}
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 OpTeX macro then their internal prefixed form is declared using _private macro. These encoding dependent declarations will be set to different values when Unicode-math font is loaded. The declared sequences for math symbols are not hyperlinked in this documentation.
The math functions like log, sin, cos are declared in the same way as in plain TeX, but they are protected in OpTeX.

These macros are defined similarly as in plain TeX. Only internal macro names from plain TeX with @ character are re-written in a more readable form.

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

Active \prime character is defined here.
\def\scalebig{\ifcase #1 0\else .6\or .72\or .9\or 1.2\or 1.5\or 1.8\else 0\fi\relax}
\protected\def\big#1{\scalebig{#1}1}
\protected\def\bigg#1{\scalebig{#1}4}
\protected\def\Big#1{\scalebig{#1}3}
\protected\def\Bigg#1{\scalebig{#1}5}
\public \big \bigg \Big \Bigg ;
\protected\def\bigl{\mathopen\big}
\protected\def\bbigl{\mathopen\bbig}
\protected\def\Bigl{\mathopen\Big}
\protected\def\bigm{\mathrel\big}
\protected\def\bbigm{\mathrel\bbig}
\protected\def\Bigm{\mathrel\Big}
\protected\def\bigr{\mathclose\big}
\protected\def\bbigr{\mathclose\bbig}
\protected\def\Bigr{\mathclose\Big}
\protected\def\biggl{\mathopen\bigg}
\protected\def\biggm{\mathrel\bigg}
\protected\def\biggr{\mathclose\bigg}
\protected\def\Biggr{\mathclose\Bigg}
\public \bigm \bigr \bbigm \bbigr ;

Math relations defined by the ~\texttt{\joinrel}~ \texttt{plain} TEX macro:
\protected\def\mathrel\joinrel{\ifx\relbar\relrelbar\gdef\mathrel\joinrel{\relbar}\fi}
\protected\def\mathrel\mathrel\relbar{\mathrel\relbar}
\ldots, \cdots, \vdots, \ddots from plain \TeX
\protected\def\ldotp{\mathinner{\ldotp\ldotp\ldotp}}
\protected\def\cdotp{\mathinner{\cdotp\cdotp\cdotp}}
\protected\def\colon{\mathinner{\colon}}
\protected\def\ldots{\mathinner{\ldotp\ldotp\ldotp}}
\protected\def\cdots{\mathinner{\cdotp\cdotp\cdotp}}
\protected\def\vdots{\vbox{\baselineskip=.4em \lineskiplimit=-\zo}}
\protected\def\ddots{\vbox{\baselineskip=.4em \lineskiplimit=-\zo}}
\protected\def\adots{\vbox{\baselineskip=.4em \lineskiplimit=-\zo}}
\math accents (encoding dependent declarations).
\protected\def\acute{\mathaccent"7013 }
\protected\def\grave{\mathaccent"7012 }
\protected\def\ddot{\mathaccent"707F }
\protected\def\tilde{\mathaccent"707E }
\protected\def\bar{\mathaccent"7016 }
\protected\def\breve{\mathaccent"7015 }
\protected\def\check{\mathaccent"7014 }
\protected\def\hat{\mathaccent"705E }
\protected\def\vec{\mathaccent"017E }
\protected\def\dot{\mathaccent"705F }
\protected\def\widetilde{\mathaccent"0365 }
\protected\def\widehat{\mathaccent"0362 }
\math, \skew, \overrightarrow, \overleftarrow, \overbrace, \underbrace macros. The last four are redefined when Unicode math is loaded.
\protected\def\moustache{\delimiter"437A340 }
\protected\def\moustache{\delimiter"537B341 }
\protected\def\lgroup{\delimiter"462833A }
\protected\def\rgroup{\delimiter"562933B }
\protected\def\arrowvert{\delimiter"26A33C }
\protected\def\Arrowvert{\delimiter"26B33D }
Macros based on \delimiter, \*witdelims and \*radical primitives.
\protected\def\bracevert{\delimiter"77C33E % the vertical bar that extends braces
\protected\def\Vert{\delimiter"26B30D } \let\|=\Vert
\protected\def\vert{\delimiter"26A30C }
\protected\def\uparrow{\delimiter"3222378 }
\protected\def\downarrow{\delimiter"3223379 }
\protected\def\updownarrow{\delimiter"326C33F }
\protected\def\Uparrow{\delimiter"322A37E }
\protected\def\Downarrow{\delimiter"322B37F }
\protected\def\Updownarrow{\delimiter"326D377 }
\protected\def\backslash{\delimiter"26E30F } % for double coset G \backslash H
\protected\def\langle{\delimiter"426830A }
\protected\def\rangle{\delimiter"526930B }
\protected\def\lbrace{\delimiter"4266308 } \let\_lbrace=\lbrace
\protected\def\rbrace{\delimiter"5267309 } \let\_rbrace=\rbrace
\protected\def\{{\ifmmode \_lbrace\else\char`{ \fi}
\protected\def\}{\ifmmode \_rbrace\else\char`}\fi}
\protected\def\rceil{\delimiter"5265307 }
\protected\def\lceil{\delimiter"4264306 }
\protected\def\rfloor{\delimiter"5263305 }
\protected\def\lfloor{\delimiter"4262304 }
\protected\def\choose{\atopwithdelims()}
\protected\def\brack{\atopwithdelims\[\]}
\protected\def\brace{\atopwithdelims\_lbrace\_rbrace}
\protected\def\_sqrt{\radical"270370 } \public \sqrt ;
\mathpalette, \vphantom, \hphantom, \phantom, \mathstrut, and \smash macros from plain \TeX.
\def\mathpalette\vphantom\hphantom\phantom\mathstrut\smash;\cong, \notin, \rightleftharpoons, \buildrel, \doteq, \bmod and \pmod macros from plain \TeX.
\def\mathpalette\vphantom\hphantom\phantom\mathstrut\smash;
The \textstyle = \scriptstyle and \scriptscriptstyle feature is new in OpTEX.

The \cases and \bordermatrix macros are almost identical as in plain \TeX. You can simply re-define \bordermatrix with other delimiters using the common \_bordermatrixwithdelims macro.

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

In \OpTeX{}, this macro is more flexible. See section 4.4 in the Typesetting Math with \OpTeX{}. The \baselineskip value is set by the \eqlines parameter and math style by the \eqstyle parameter.
The \texttt{displaylines} command creates horizontally centered formulae. It behaves exactly as in Plain TeX. The \texttt{halign} is applied directly in the outer display environment with lines of type \texttt{ hbox to displaywidth}. This enables to break lines inside such display to more pages but it is impossible to use \texttt{eqno} or \texttt{leqno} or \texttt{eqmark}.

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

These macros are inspired by \texttt{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, \not\XXX is done in the usual way.

\protected\def \notchar "3236
\protected\def \_not#1{%
  \ifx #1< \_nless \else
  \ifx #1> \_ngtr \else
    \edef \_tmpn{\csstring#1}%
    \ifcsname _not\_tmpn\endcsname \csname _not\_tmpn\endcsname
    \else \ifcsname _n\_tmpn\endcsname \csname _n\_tmpn\endcsname
    \else \mathrel{\mathord{\notchar}\mathord{#1}}%\fi\fi\fi\fi}
\private
\_mathchardef \_notchar "3236
\_protected\def \_notchar "3236
\protected\def \_not#1{%
  \ifx #1< \_nless \else
  \ifx #1> \_ngtr \else
    \edef \_tmpn{\csstring#1}%
    \ifcsname _not\_tmpn\endcsname \csname _not\_tmpn\endcsname
    \else \ifcsname _n\_tmpn\endcsname \csname _n\_tmpn\endcsname
    \else \mathrel{\mathord{\notchar}\mathord{#1}}%\fi\fi\fi\fi}
\private
\nleq \ngeq \nless \ngtr \nprec \nsucc \nleqslant \ngeqslant \preceq
\nsucceq \nleqslant \ngeqslant \sim \cong \subset \supset \subseteq \supseteq
\subseteq \supseteq \parallel \mid \shortmid \shortparallel \vdash \Vdash
\vDash \VDash \trianglerighteq \trianglelefteq \triangleright \leftarrow \rightarrow \Leftarrow \Rightarrow
\Leftrightarrow \leftrightarrow \exists
\private
\mathstyles{\langle\math list\rangle} behaves like \{\langle\math list\rangle\}, but you can use the following commands in the \langle\math list\rangle:

- \currystyle which expands to \displaystyle, \textstyle, \scriptstyle or \scriptscriptstyle depending on the current math style when \mathstyles was opened.
- \dobystyle{\{D\}}{\{T\}}{\{S\}}{\{SS\}} is expandable macro. It expands to \{D\}, \{T\}, \{S\} or \{SS\} depending on the current math style when \mathstyles was opened.
- \stylenum is 0, 1, 2 or 3 depending on the current math style when \mathstyles was opened.

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

\mathstyles{\langle\math list\rangle} behaves like \{\langle\math list\rangle\}, but you can use the following commands in the \langle\math list\rangle:

- \currystyle which expands to \displaystyle, \textstyle, \scriptstyle or \scriptscriptstyle depending on the current math style when \mathstyles was opened.
- \dobystyle{\{D\}}{\{T\}}{\{S\}}{\{SS\}} is expandable macro. It expands to \{D\}, \{T\}, \{S\} or \{SS\} depending on the current math style when \mathstyles was opened.
- \stylenum is 0, 1, 2 or 3 depending on the current math style when \mathstyles was opened.

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

\mathstyles{\langle\math list\rangle} behaves like \{\langle\math list\rangle\}, but you can use the following commands in the \langle\math list\rangle:

- \currystyle which expands to \displaystyle, \textstyle, \scriptstyle or \scriptscriptstyle depending on the current math style when \mathstyles was opened.
- \dobystyle{\{D\}}{\{T\}}{\{S\}}{\{SS\}} is expandable macro. It expands to \{D\}, \{T\}, \{S\} or \{SS\} depending on the current math style when \mathstyles was opened.
- \stylenum is 0, 1, 2 or 3 depending on the current math style when \mathstyles was opened.

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

\mathstyles{\langle\math list\rangle} behaves like \{\langle\math list\rangle\}, but you can use the following commands in the \langle\math list\rangle:

- \currystyle which expands to \displaystyle, \textstyle, \scriptstyle or \scriptscriptstyle depending on the current math style when \mathstyles was opened.
- \dobystyle{\{D\}}{\{T\}}{\{S\}}{\{SS\}} is expandable macro. It expands to \{D\}, \{T\}, \{S\} or \{SS\} depending on the current math style when \mathstyles was opened.
- \stylenum is 0, 1, 2 or 3 depending on the current math style when \mathstyles was opened.

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

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

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

\mathstyles{saves current math style (including its cramped/normal subversion) and \usemathstyle restores the saved math style. These macros are based on the LuaTeX's \mathstyle primitive, i.e. they don't work in generalized fractions.
Usage: \def\mathclap #1{{\usemathstyle \hbox toOpt{\hss\usemathstyle#1\hss}}}.
The \texttt{\mathbox{⟨text⟩}} macro is copied from OPmac trick 078. It behaves like \texttt{\hbox{⟨text⟩}} but the ⟨text⟩ is scaled to a smaller size if it is used in 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.

\texttt{\loadmath \{{Unicode-math font}\}} macro loads math fonts and redefines all default math-codes using \texttt{\input unixmath-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 ⟨normal-font⟩ was sucessfully loaded by the previous \texttt{\loadmath}. For example:

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

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 main math font is loaded by \texttt{\loadmath} (typically indirectly using \texttt{\fontfam}) and you can load more additional math fonts by \texttt{\addUmathfont}:

\begin{verbatim}
\addUmathfont \famname \{{[normal-font]\}} \{{\ffeatures\}} \{{[bold-font]\}} \{{\ffeatures\}} \{{\factor\}}
\end{verbatim}

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

\begin{verbatim}
\fontfam[1m] \% does \lodmath\{{[latinmodern-math]\}}
\addUmathfont \xits \{{[XITSMath-Regular]\}} \{{[XITSMath-Bold]\}}
\end{verbatim}

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

All characters used in math formula are printed from main math font by default. But you can re-define characters for printing from additional font by \texttt{\mathchars \famname \{{list of sequences\}}}. For example:

\begin{verbatim}
\mathchars \xits \{{\stareq \triangleq \veeeq \wedgeq\}}
\end{verbatim}

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

The \texttt{\mathchars} macro keeps the class and slot of declared math objects and re-declares only family of them. It is applied to all control sequences given in the parameter. The relevant math codes are re-declared.
Use \addto\selector{\fam\famname} if you want to print whole math alphabet from an additional math font. For example \addto\cal{\fam\xits} declares all \cal characters from the \xits font loaded by \addUmathfont.

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

The \mathcodes macro declares mathcode of given characters internally by

\mathcode{⟨character⟩} = ⟨class-number⟩⟨family⟩{⟨character⟩}

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

2.16.1 Unicode-math macros preloaded in the format

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

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

\noloadmath disallows Unicode-math loading by \mathfalse.
\doloadmath allows Unicode-math loading by \mathtrue.

\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 emblodn LuaTeX font feature.
The Unicode version of the \normalmath and \boldmath macros are defined here as \normalunimath and \boldunimath macros. They are using \setunimathdimens in a similar sense as \setmathdimens. You can combine more fonts if you register them to another math families (5, 6, 7, etc.) in the \normalmath macro.

The default value of \normalunimath shows a combination of base Unicode-math font at family 1 with 8bit Math font at family 4. See definition of \script macro where \fam4 is used.

\def\normalunimath{%
\setmathfamily 0 \tenrm % font for non-math objects in math mode
\loadumathfamily 1 {\unimathfont}{} % Base font
\loadmathfamily 4 rsfs % script
\setunimathdimens
}%
\def\boldunimath{%
\setmathfamily 0 \tenbf % font for non-math objects in math mode
\ifx\unimathboldfont\undefined
\loadumathfamily 1 {\unimathfont}{embolden=1.7;} % Base faked bold
\else
\loadumathfamily 1 {\unimathboldfont}{} % Base real bold font
\fi
\loadmathfamily 4 rsfs % script
\setunimathdimens
}%
\def\setunimathdimens{% PlainTeX sets these dimens for 10pt size only:
\delimitershortfall=0.5\fontdimen6\textfont1
\nulldelimiterspace=0.12\fontdimen6\textfont1
\parindent=0.12\fontdimen6\textfont1
\setbox0=\hbox{\everymath{}$\fam1\displaystyle{0\atop0}$} % Umathfractiondelsize
\Umathfractiondelsize\displaystyle = \dimexpr(\ht0-\Umathaxis\displaystyle)*2\relax
\setbox0=\voidbox}

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

\begin{verbatim}
\loadmath[{xitsmath-regular}]
\def\boldmath{%
\loadumathfamily 1 {[xitsmath-bold]}{} % Base font
\loadmathfamily 4 rsfs % script
\loadumathfamily 5 {[xitsmath-regular]}{}
\def\{|{\Udelimiter 0 5 "02016 }% % norm delimiter from family 5
\setunimathdimens
}
\end{verbatim}
\addUmathfont ⟨fam⟩ {⟨normal-font⟩} {⟨features⟩} {⟨bold-font⟩} {⟨features⟩} {⟨factor⟩} allocates new \textup{fam} using \newfam and adds loading this font to the \normalmath and \boldmath macros. Note that allocationos using \newfam starts from 43 because numbers 1–42 are reserved for direct usage without \newfam. We use \aheadto here because we want to read the main family 1 as last one (for definitive setting of math parameters).

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

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

\Umathchardef ⟨sequence⟩ = ⟨math class⟩ ⟨math family⟩ ⟨slot number⟩
\Umathcode ⟨code⟩ = ⟨math class⟩ ⟨math family⟩ ⟨slot number⟩
\Udelcode ⟨code⟩ = ⟨math family⟩ ⟨slot number⟩
\mathcodes {\texttt{\textbackslash family}} \{\{\textit{list of pairs}\}\} sets mathcodes of given characters with explicit \texttt{\textbackslash class}es. Each pair can be \texttt{\textbackslash class}\{\{\textit{list of chars}\}\} and \{\textit{list of chars}\} can include \texttt{\textbackslash Urange \{from\} \{to\}}. This is reason why we apply \texttt{\textbackslash expanded} to the \{\textit{list of chars}\} before reading it by \texttt{\textbackslash foreach}: the \texttt{\textbackslash Urange} is expandable and expands to the relevant list of characters.

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

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

Unicodė math font includes all typical math alphabets together, user needs no load more \TeX math families. These math alphabets are encoded by different parts of Unicode table. We need auxiliary macros for setting mathcodes by selected math alphabet.

\texttt{\textbackslash umathrange \{\{\textit{from}\} \{\textit{to}\}\} \{\textit{class}\} \{\textit{family}\} \{\textit{first}\}} sets \texttt{\textbackslash mathcodes} of the characters in the interval \{\textit{from}\} \{\textit{to}\} to \{\textit{first}\}, \{\textit{first}\}+1, \{\textit{first}\}+2 etc., but \texttt{\textbackslash umathcharholes} are skipped (\texttt{\textbackslash umathcharholes} are parts of the Unicode table not designed for math alphabets, they cause that the math alphabets are not continuously spread out in the table; I mean that the designers were under the influence of drugs when they created this part of the Unicode table). The \{\textit{from}\} \{\textit{to}\} clause includes characters like A–Z. Note that the \texttt{\textbackslash umathrange} sets the \texttt{\textbackslash classfam} macro as \{\textit{class}\} \{\textit{family}\} for later use.

A few math characters have very specific Unicode and must be handled individually. We can run \texttt{\textbackslash umathrangespec} \{\textit{list of characters}\} \texttt{\textbackslash relax} just after \texttt{\textbackslash umathrange}. The \texttt{\textbackslash umathnumB} must be set to the first destination code. The \texttt{\textbackslash umathrangespec} applies to each character from the \{\textit{list of characters}\} this: \texttt{\textbackslash Umathcode \{char-code\} \{\texttt{\textbackslash classfam} \texttt{\textbackslash umathnumB}\} \texttt{\textbackslash increment} \texttt{\textbackslash umathnumB}}. If \texttt{\textbackslash umathnumB}=0 then it applies \texttt{\textbackslash Umathcode \{char-code\} \{\texttt{\textbackslash classfam} \texttt{\textbackslash umathnumB}\}}. The \texttt{\textbackslash classfam} and \texttt{\textbackslash umathnumB} were typically set by previous call of the \texttt{\textbackslash umathrange} macro.
The math alphabets are set by \_rmvariables, \_bfvariables, \_itvariables, \_bivariables, \_calvariables, \_bcalvariables, \_frakvariables, \_bfrakvariables, \_bbvariables, \_sansvariables, \_bsansvariables, \_sansgreekvariables, \_bsansgreekvariables, \_ttdigits, \_ttvariables, \_itgreek, \_rmgreek, \_bfgreek, \_bigreek, \_bsangreekvariables, \_bsangreekvariables, \_greekvariables, \_greekvariables. They are declared using the \_umathrange{⟨range⟩}{⟨class⟩}{⟨family⟩}{⟨starting-code⟩} macro.

\_chardef\_ncharrmA=`A \_chardef\_ncharrma=`a
\_chardef\_ncharbfA=`1D400 \_chardef\_ncharbfa=`1D41A
\_chardef\_ncharitA=`1D434 \_chardef\_ncharita=`1D44E
\_chardef\_ncharbiA=`1D468 \_chardef\_ncharbia=`1D482
\_chardef\_ncharclA=`1D49C \_chardef\_ncharcla=`1D4B6
\_chardef\_ncharbcA=`1D4D0 \_chardef\_ncharbca=`1D4EA
\_chardef\_ncharfrA=`1D504 \_chardef\_ncharfra=`1D51E
\_chardef\_ncharbrA=`1D56C \_chardef\_ncharbra=`1D586
\_chardef\_ncharbbA=`1D538 \_chardef\_ncharbba=`1D552
\_chardef\_ncharsnA=`1D5A0 \_chardef\_ncharsna=`1D5BA
\_chardef\_ncharbsA=`1D5D4 \_chardef\_ncharbsa=`1D5EE
\_protected\_def\_greekrmA="0391 \_chardef\_greekrmA="03B1
\_chardef\_greekbfA="1D6A8 \_chardef\_greekbfa="1D6C2
\_chardef\_greekbiA="1D6E2 \_chardef\_greekbia="1D6F7
\_chardef\_greeksnA="1D756 \_chardef\_greeksna="1D770
\_chardef\_greeksiA="1D790 \_chardef\_greeksia="1D7AA
\_protected\_def\_itgreek \_umathrangegreek\_greekita
\_protected\_def\_rmgreek \_umathrangegreek\_greekrma
\_protected\_def\_bfgreek \_umathrangegreek\_greekbfa
\_protected\_def\_bigreek \_umathrangegreek\_greekbia
\_protected\_def\_bsangreek \_umathrangegreek\_greekbsa
\_protected\_def\_bbangreek \_umathrangegreek71\_greekbsa
\_protected\_def\_itGreek \_umathrangeGREEK\_greekitaA
\_protected\_def\_rmGreek \_umathrangeGREEK\_greekrmaA
\_protected\_def\_bfgreek \_umathrangeGREEK\_greekbfaA
\_protected\_def\_bigreek \_umathrangeGREEK\_greekbiaA
\_protected\_def\_bsangreek \_umathrangeGREEK\_greekbsaA
\_protected\_def\_bbangreek \_umathrangeGREEK\_greekbsaA
\_protected\_def\_digitrmO=`0 \_chardef\_digitrmO=`0
\_chardef\_digitbfO="1D7CE
\_chardef\_digitbbO="1D7D8
\_chardef\_digitbsO="1D7E2
\_chardef\_digitttO="1D7EC
\_protected\_def\_rmdigits \_umathrange{0-9}\_digitrmO
\_protected\_def\_bfdigits \_umathrange{0-9}\_digitbfO
\_protected\_def\_bbdigits \_umathrange{0-9}\_digitbbO
The control sequences for \alpha, \beta, etc. are redefined here. The \alpha will expand to the character with Unicode “\U03B1”, this is a normal character \(\alpha\). You can type it directly in your editor if you know how to do this. These sequences are declared by \_greekdef{list of sequences}\relax.

The \_umathrangeGREEK{class}{family}{first} and \_umathrangegreek{class}{family}{first} macros for setting math codes of Greek characters are defined here. They use \_umathrange for general codes but the exceptions must be handled by the \_umathrangespec macro. The exceptions are seven Greek characters: \(\epsilon, \vartheta, \varkappa, \phi, \varrho, \varpi, \nabla\). The first six of these characters should behave as lowercase Greek letters and the last one \nabla is uppercase Greek letter.

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

You can redefine them again if you need different behavior (for example you don’t want to use sans serif bold in math). What to do:

\_protected\def\_mabf {\_inmath{\_bfvariables\_bfgreek\_bfGreek\_bfdigits}}
\_protected\def\_mabi {\_inmath{\_bivariables\_bigreek\_bfGreek\_bfdigits}}
\_protected\def\_matt {\_inmath{\_ttvariables\_ttdigits}}

\_inmath {\cmds} applies {\cmds} only in math mode.

% You can redefine these macros to follow your wishes.
% For example, you need upright lowercase greek letters, you don’t need % \bf and \bi behave as sans serif bold in math, ...
\_protected\def\_marm {\_inmath{\_rmvariables \_rmdigits}}
\_protected\def\_mait {\_inmath{\_tvariables \_ttGreek}}
\_protected\def\_mabi {\_inmath{\_bsansvariables \_bsansgreek \_bsansGreek \_bsansdigits}}
\_protected\def\_matt {\_inmath{\_ttvariables \_ttdigits}}

% to keep off \loop processing in text mode
\_protected\_inmath\#1\_relax
Each Unicode slot carries information about math type. This is saved in the file MathClass-15.txt which is copied to mathclass.opm The file has the following format:

```
mathclass.opm
```

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

```
unimath-codes.opm
```

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

```
unimath-table.opm
```

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

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

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

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

Many special characters must be declared with care...

\begin{verbatim}
\begingroup % input unimath-table-0pm (it is a copy of unicode-math-table.tex):
\_umathnumB="F800 % pointer to the Private User Area
\_edef\UmathSymbol#1#2#3#4{%
  \edef#2{\Uchar #1}% control sequence is a macro which expands to the Unicode character
  \ifnum#1=\Umathcodenum#1 \Umathcode#1=0 1 #1 \fi % it isn't set by mathclass.0pm
  \ifx#3\mathaccent \protected\def#2{\Umathaccent fixed 7 1 #1 }\fi
  \ifnum#1>"390 \ifnum#1<"3F6
    \edef#2{\Uchar\umathnumB}% \mupAlpha, \mupBeta, \mupalpha, \mupbeta, ...
    \Umathcode\umathnumB=0 1 #1
    \advance\umathnumB by1
  \fi\fi % \muGreek, \mugreek symbols
}\def\mathfence{F}%
globaldefs=1 \_input unimath-table.0pm
\endgroup
\end{verbatim}
Aliases are declared here. They are names not mentioned in the `unimath-table.opm` file but commonly used in TeX.
The \texttt{\textbackslash not} macro is redefined here. If the \texttt{\textbackslash not!⟨char⟩} is defined (by \texttt{\negationof}) then this macro is used. Else centered / is printed over the ⟨char⟩.

\begin{verbatim}
\protected\def\not#1{%\trycs{not!\csstring#1}{\mathrel{\mathstyles{%\setbox0=\hbox{\math\currstyle#1}\hbox to \wd0{\hspace{-\wd0}\box0}\kern-\wd0 \box0}}}\def\negationof#1#2{\ea\let\csname _not!\csstring#1\endcsname=#2}}
\negationof\neq\neq\neg\less
\negationof\ngtr\rangle\ngtr\forks\forks
\negationof\in\notin\mid\nmid\cong\ncong\leftarrow\nleftarrow\rightarrow\nrightarrow\end{verbatim}
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.

Public \not ;

2.16.3 More Unicode-math examples

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


2.16.4 Printing all Unicode math slots in used math font

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

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

\input print-unimath.opm
2.17 Scaling fonts in document (high-level macros)

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

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

\texttt{\textbackslash typoscale \{font-factor/\textbackslash baseline-factor\}} scales font size and baselineskip by given factors in respect to current values. It calculates the \texttt{\textbackslash typosize} parameters and runs the \texttt{\textbackslash typosize}. 108
\_protected\_def \_typoscale [\#1/\#2]\{
  \_if$\#1$\_def \_tmp{[/}\_else
  \_settmpdim{\#1}\_optsize
  \_edef \_tmp{[\_ea\_ignorept\_the\_\_tmpdim]}\_fi
  \_if$\#2$\_edef \_tmp{\_tmp]\_else
  \_settmpdim{\#2}\_baselineskip
  \_edef \_tmp{\_tmp \_ea\_ignorept\_the\_\_tmpdim]}\_fi
  \_ea\_typosize \_tmp
}\}
\_def \_settmpdim\#1\#2{\_tmpdim=\#1pt \_divide \_tmpdim by\_1000
  \_tmpdim=\_ea\_ignorept\_the\_\_tmpdim
}\_public \_typoscale ;
\_setbaselineskip \{\langle\_baselineskip\rangle\} \sets new\_baselineskip \and more values of registers which are dependent on the \langle\_baselineskip\rangle including the \strutbox.
\_setmainvalues \sets the current font size and \baselineskip values to the \mainforsize and \mainbaselineskip registers and loads fonts at given sizes. It redefines itself as \_setmainvaluesL to set the main values only first. The \_setmainvaluesL does only fonts loading. \scalemain \returns to these values if they were set. Else they are set to 10/12pt. \mfontsrule \gives the rule how math fonts are loaded when \textsize or \typoscale are used. The value of \mfontsrule can be:
\begin{itemize}
  \item 0: no math fonts are loaded. User must use \textmath or \boldmath explicitly.
  \item 1: \textnormalmath is run if \textsize or \typoscale are used first or they are run at outer group level. No \everymath/\everypdisplay are set in this case. If \textsize or \typoscale are run repeatedly in a group then \textnormalmath is run only when math formula occurs. This is done using \everymath/\everydisplay and \textmath. \mfontsrule=1 is default.
  \item 2: \textnormalmath is run whenever \textsize or \typoscale are used. \everymath/\everypdisplays are untouched.
\end{itemize}
\let \setmainvaluesL \_setmainvalues
\fi
\optsize=\mainfoslsize \baselineskip=\mainbaselineskip
}

\public \scalemain \mainfoslsize \mainbaselineskip \mfontsrule ;

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

\def \setmathfonts\$\normalmath\everymath{}\everydisplay{}$

\thefontsize \[\langle \text{size} \rangle \]
and \thefontscale \[\langle \text{factor} \rangle \]
do modification of the size of the current font. They are implemented by the \newcurrfontsize macro.

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

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

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

```
\_def \rmfixed \% used in default \footline
\_ifdim \mainfonsize=0pt \_mainfonsize=10pt \_fi
\_fontdef \tenrm \_setfontsize{at \mainfonsize} \_resetmod \rm
\_global \let \rmfixed = \tenrm \%
\_rmfixed
\_let \rmfixed = \tenrm \%
```

2.18 Output routine

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

- \_begoutput which does:
  - increments \gpage,
  - prints \_Xpage{\langle gpage \rangle} to the .ref file (if \openref is active),
  - calculates \_hoffset,
  - sets local meaning of macros used in headlines/footlines (see \_regmacro).
- \_shipout \_completepage, which is \_vbox of –
  - background box, if \pgbackground is non-empty,
  - headline box by \_makeheadline, if the \_headline is nonempty,
  - \_vbox to \_size of \_pagecontents which consists of –
    - \_pagedest, the page destination \_pg\langle gpage \rangle for hyperlinks is created here,
    - \topins box if non-empty (from \_topinserts),
    - \_box255 with completed vertical material from main vertical mode,
    - \_footnoterule and \_footins box if nonempty (from \_fnote, \_footnote),
    - \_pgbottomskip (default is 0 pt).
  - footline box by \_makefootline, if the \_footline is nonempty
- \_endoutput which does:
  - increments \gpage using \_advancepageno
  - runs output routine repeatedly if \dosupereject is activated.

```
\_optexoutput is the default output routine. You can create another
```

```
\_optexoutput={\_optexoutput}
\_def \_optexoutput{\_begoutput \_optexshipout \_completepage \_endoutput}
```

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.

```
\_def \_begoutput{\_incr \_gpage}
\_immediate \wref \_Xpage{\langle \_the \_gpage \rangle \langle \_folio \rangle}\
\_setxhsize \_prepoffsets \_the \_regmark
\_def \_endoutput{\_advancepageno
\_setxhsize \_prepoffsets \_themark}
```

The \_optexshipout does similar work like the \_shipout primitive. The color literals are added to the \_box0 using the \_preshipout{destination box number}{box specification} pseudo-primitive. It is defined using lua code, see section 2.39. Finally the \_shipout primitive is used. We want to use \_hoffset value
and not \pagerightoffset during shipout, so the \pagedir and \box0 are set to TLT. If a user sets different value of \pagedir in the document, it is irrelevant for this \shipout process.

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.

\_makeheadline creates \vbox to0pt with its contents (the \headline) shifted by \headlinedist up.

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

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

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

\begin{verbatim}
\_countdef\_pageno=0 \_pageno=1 % first page is number 1
\_def \_folio {\_ifnum\_pageno<0 \_romannumeral-\_pageno \_else \_number\_pageno \_fi}
\_def \_nopagenumbers {\_footline={}}
\_def \_advancepageno {\_ifnum\_pageno<0 \_decr\_pageno \_else \_incr\_pageno \_fi}% increase |pageno|
\_def \_raggedbottom {\_topskip=\_dimexpr\_topskip plus60pt \_pgbottomskip=0pt plus1fil\relax}
\_def \_normalbottom {\_topskip=\_dimexpr\_topskip\_pgbottomskip=0pt\relax}
\_public \pageno \folio \nopagenumbers \advancepageno \raggedbottom \normalbottom;
\end{verbatim}

Macros for footnotes are the same as in \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 the \fnote macro is used. The \opfootnote nor \vfootnote don’t take the footnote text as a parameter. This is due to a user can do catcode settings (like inline verbatim) in the footnote text. This idea is adapted from plain \TeX{}. The \footnote and \footstrut is defined as in plain \TeX{}.

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

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

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

\begin{verbatim}
\def \draft {\texttt{\textbackslash pgbackground=\{\texttt{\textbackslash draftbox{\{\texttt{\textbackslash draftfont DRAFT}\}}}}\}
\global \let \draftfont = \draftfont
\public \draft
\end{verbatim}

\section{Margins}

The \texttt{\textbackslash margins} macro is documented in the section 1.2.1.

\begin{verbatim}
\def \margins/#1#2 (#3,#4,#5,#6)#7 {
  \if\temp{\empty}
    \opwarning{\texttt{\textbackslash margins: missing unit, mm inserted}}
  \else
    \setpagedims #2 % setting \texttt{\textbackslash pgwidth}, \texttt{\textbackslash pgheight}
    \ifdim \pgwidth=0pt \else
      \hoffset=0pt \voffset=0pt
      \if#3 \hoffset = \dimexpr (\pgwidth - \hsize)/2 \relax % only right margin
      \else \hoffset = \dimexpr \pgwidth - \hsize - #3 \relax % only left margin
      \fi
      \else \if#5 \hoffset = \dimexpr \pgwidth - #5 \relax % left+right margin
      \else \hsize = \dimexpr \pgwidth - \hsize - #5 \relax % left margin
      \fi
      \fi
      \fi
      \if#4 \voffset = \dimexpr (\pgheight - \vsize)/2 \relax % only top margin
      \else \voffset = \dimexpr \pgheight - \vsize - #4 \relax % only bottom margin
      \fi
      \else \if#6 \voffset = \dimexpr \pgheight - #6 \relax % top+bottom margin
      \else \vsize = \dimexpr \pgheight - \vsize - #6 \relax % top margin
      \fi
      \fi
      \fi
      \fi
      \if\temp{\empty}
    \else
      \ifodd \pageno \else \advance \hoffset \shiftoffset \fi
    \fi
  \fi
  \setpagedimsC \pgwidth=#1:#3
  \setpagedimsC \pgheight=#2:#5
  \pdfpagewidth=\pgwidth \pdfpageheight=\pgheight
}
\def \setpagedims{
  \ifnextchar({\setpagedimsB}{\setpagedimsA}
\def \setpagedimsA#1 {
  \ifcsname _pgs:#1\endcsname
    \ea \ea \ea \setpagedimsB \csname _pgs:#1\ea \endcsname
  \else
    \opwarning{page specification "#1" is undefined} \fi
}
\def \setpagedimsB (#1,#2)#3 {
  \setpagedimsC \pgwidth=#1:#3
  \setpagedimsC \pgheight=#2:#3
  \pdfpagewidth=\pgwidth \pdfpageheight=\pgheight
}
\end{verbatim}

\section{Margins (pg)}

(margin, left, right, top, bot, unit) takes its parameters, does calculation and sets \texttt{\textbackslash hoffset}, \texttt{\textbackslash voffset}, \texttt{\textbackslash hsize} and \texttt{\textbackslash 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.

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

\_public \margins ;

\_sdef{_pgs:a3}{(297,420)mm} \_sdef{_pgs:a4}{(210,297)mm} \_sdef{_pgs:a5}{(148,210)mm}

\_sdef{_pgs:a3l}{(420,297)mm} \_sdef{_pgs:a4l}{(297,210)mm} \_sdef{_pgs:a5l}{(210,148)mm}

\_sdef{_pgs:b5}{(176,250)mm} \_sdef{_pgs:letter}{(8.5,11)in}

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

\_def\_trueunit{}

\_def\_magscale[#1]{\_mag=#1\_def\_trueunit{true} \_ifdim\_pgwidth=0pt \_else \_truedimen\_pgwidth \_truedimen\_pgheight \_fi \_truedimen\_pdfpagewidth \_truedimen\_pdfpageheight}

\_def\_truedimen#1{\_ifx\_trueunit\_empty \_else#1=\_ea\_ignorept\_the#1truept \_fi}

\_public \magscale ;

\section{2.20 Colors}

\subsection{2.20.1 Basic concept}

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

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

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

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

Macros \texttt{\setcmykcolor{⟨C⟩ ⟨M⟩ ⟨Y⟩ ⟨K⟩}} or \texttt{\setrgbcolor{⟨R⟩ ⟨G⟩ ⟨B⟩}} or \texttt{\setgreycolor{⟨Grey⟩}} are used in color selectors. These macros expand to internal \texttt{\_setcolor} macro which sets the \texttt{\_colorattr} attribute to an integer value and prepares mapping between this value and the real color data. This mapping is used just before each \texttt{\shipout} in output routine. The \texttt{\_preshipout} pseudo-primitive is used here, it converts attribute values to internal PDF commands for selecting colors.

\subsection{2.20.2 Color mixing}

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

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

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

115
The \texttt{\useK} command exactly does:

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

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

\texttt{\colordef \Color {\Brown-\Black}}

can be used for removing the black component from the color. You can use the \texttt{-\Black} trick after \texttt{\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.

\texttt{\colordef\mycolor{\Grey+.6^\Blue} % the same as \colordef\mycolor{\Grey+.6\Yellow}}

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

If a CMYK to RGB or RGB to CMYK conversion is needed then direct conversion of given color is used (if declared using \texttt{\rgbcmykmap(\rgb{\langle cmyk \rangle})}) or 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 \texttt{\setrgbcolor} and it is used in \texttt{\colordef} or if it is printed when \texttt{\onlycmyk} is declared. The CMYK to RGB conversion is invoked when a color is declared using \texttt{\setcmykcolor} and it is used in \texttt{\rgbcolordef} or if it is printed when \texttt{\onlyrgb} is declared.

### 2.20.3 Implementation

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

By default, the \texttt{\setcmykcolor} \texttt{\setrgbcolor} and \texttt{\setgreycolor} macros with \texttt{(\langle components\rangle)} parameter expand to \texttt{\setcolor(\langle color-data\rangle)\{\langle fill-op\rangle\}\{\langle stroke-op\rangle\}} where \texttt{\langle color-data\rangle} is \texttt{\langle R \rangle \langle G \rangle \langle B \rangle} or \texttt{\langle C \rangle \langle M \rangle \langle Y \rangle \langle K \rangle} or \texttt{\langle G \rangle} and \texttt{\langle fill-op\rangle} is color operator for filling, \texttt{\langle stroke-op\rangle} is color operator for stroking.

The \texttt{\onlyrgb} declaration redefines \texttt{\setcmykcolor} to do conversion to RGB just before \texttt{\setcolor} is used. The \texttt{\onlycmyk} declaration redefines \texttt{\setrgbcolor} to do conversion to CMYK just before
\_setcolor is used. Moreover, \onlyrgb re-defines three basic RGB colors for RGB color space and re-declares \colordef as \rgbcolordef.

\begin{verbatim}
colors.opm
\def\onlyrgb{\def\Red{\setrgbcolor{1 0 0}}\def\Green{\setrgbcolor{0 1 0}}\def\Blue{\setrgbcolor{0 0 1}}\let\colordef=\rgbcolordef
\def\setrgbcolor##1{\setcolor{##1}{rg}{RG}}\def\setcmykcolor##1{\ea\setcolor\ea{\expanded{\cmyktorgb ##1 ;}}{rg}{RG}}
\public \colordef \setrgbcolor \setcmykcolor ;}
\def\onlycmyk{\let\colordef=\cmykcolordef\def\setrgbcolor##1{\ea\setcolor\ea{\expanded{\rgbtocmyk ##1 ;}}kK}\def\setcmykcolor##1{\setcolor{##1}kK}\public \colordef \setrgbcolor \setcmykcolor ;}
\public \onlyrgb \onlycmyk ;
\end{verbatim}

The \colorattr for coloring is allocated and \setcolor{⟨color-data⟩}{⟨fill-op⟩}{⟨stroke-op⟩} is defined here. This macro does \colorattr=\colorcnt if the ⟨color data⟩ was not used before and prepare mapping from this integer value to the ⟨color data⟩ and increments \colorcnt. If the ⟨color data⟩ were used already, then \setcolor does \colorattr=⟨stored-value⟩. This work is done by the \translatecolor macro. The following mapping macros are created:

\begin{verbatim}
\color::⟨data⟩ ⟨fill-op⟩ ... expands to used ⟨attribute-value⟩
\color:⟨attribute-value⟩ ... expands to ⟨data⟩ ⟨fill-op⟩
\color-s:⟨attribute-value⟩ ... expands to ⟨data⟩ ⟨stroke-op⟩
\end{verbatim}

\begin{verbatim}
colors.opm
\def\transparency{\afterassignment\transparencyA \transpattr}
\def\transparencyA{\ifnum\transpattr<1 \transpattr=\noattr \fi\ifnum\transpattr>255 \noexpand\transparency > 255 not allowed\transpattr=\noattr \else\edef\transpv{\expr{(255-\the\transpattr)/255}}\fi}
\def\nolocalcolor{\let\colorprefix=\relax}
\def\localcolor{\let\colorprefix=\global}
\def\localcolor{\let\colorprefix=\global}
\public \localcolor \nolocalcolor ;
\end{verbatim}

We support concept of non-local color, i.e. all changes of the color attribute are global by setting \colorprefix to \global. \localcolor is the default, i.e. \colorprefix is \relax.

You can write \global\Red if you want to have global setting of the color.

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

\resetattrs resets the \colorattr and \transpattr to their initial value −"7FFFFFFF. 

117

\begin{verbatim}
\newattribute \transpattr
\def\transparency{\afterassignment\transparencyA \transpattr}
\def\transparencyA{\ifnum\transpattr<1 \transpattr=\noattr \fi\ifnum\transpattr>255 \noexpand\transparency > 255 not allowed\transpattr=\noattr \else\edef\transpv{\expr{(255-\the\transpattr)/255}}\fi}
\def\nolocalcolor{\let\colorprefix=\relax}
\def\localcolor{\let\colorprefix=\global}
\def\localcolor{\let\colorprefix=\global}
\public \localcolor \nolocalcolor ;
\end{verbatim}

117
We use Lua codes for RGB to CMYK or CMYK to RGB conversions and for addition color components in the `colordef` macro. The `\_rgbtocmyk` \{R \} \{G \} \{B \} expands to \{C \} \{M \} \{Y \} \{K \} and the `\_cmyktorgb` \{C \} \{M \} \{Y \} \{K \} expands to \{R \} \{G \} \{B \}. The `\_colorcrop`, `\_colordefFin` and `\_douseK` are auxiliary macros used in the `colordef`. The `\_colorcrop` rescales color components in order to they are in \([0, 1]\) interval. The `\_colordefFin` expands to the values accumulated in Lua code `color_C`, `color_M`, `color_Y` and `color_K`. The `\_douseK` applies `\useK` to CMYK components. The `\_tocmyk` \{rgb\} or `\_torgb` \{cmyk\} control sequences (given by `\rgbcmykmap`) have precedence.

We have a problem with the `%3f` directive in Lua code. It prints trailed zeros: (0.300 instead desired 0.3) but we want to save PDF file space. The macro `\_stripzeros` removes these trailing zeros at the expand processor level. So `\_stripzeros 0.300 0.400 0.560 ;` expands to `.3 .4 .56`.
\rgbcolordef \{R\} \{G\} \{B\} \{(C)\} \{(M)\} \{(Y)\} \{(K)\} declares mapping from RGB to CMYK and from CMYK to RGB for given color. It has precedence before general formulae used in the \rgbtocmyk and \cmyktorgb macros. Note, that the values \{R\} \{G\} \{B\} \{(C)\} \{(M)\} \{(Y)\} \{(K)\} must be given exactly in the same format as in \setcmykcolor and \setrgbcolor parameters. For example, 0.5 or .5 or .50 are different values from point of view of this mapping.

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

```plaintext
input \{(data)\}: \".3\Blue + .6\KhakiC \useK -\Black\"
expanded to: \".3 \nekwk \{1 1 0 0 \usek \} \{+ 0.6\} \{\usek \} \{= k\} 0.804 0.776 0.45 \usek \{= k\} 0 0 0 0 \usek\"
adjusted to: \"_addcolor .3! =K 1 1 0 0 _addcolor .6! ^R .804 .776 .45 _useK -!G 0" and this is processed.
```

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

Next, \commoncolordef saves resulting values from Lua to \tmpb using \colordefFin. If \rgbcolordef is processed, then we must to remove the last \{(K)\} component which is in the format .0 in such case. The \stripK macro does it. Finally, the \what-define is defined as \what{\{(selector)\} \{(expanded_\tmpb)\}} 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\%
\directlua{color_C=0 color_M=0 color_Y=0 color_K=0}
\begingroup
\edef\tmpb{\cmykcolordef \{(\#1 \#2 \#3 \#4 \#5)\}}
\ifx \#2 \edef\tmpb{\cmykcolordef \{(\#1 \#2 \#3 \#4 \#5)\}}
\let\useK=\relax
\edef\tmpb{\addcolor ! \{(\#1 \#2 \#3 \#4 \#5)\}}
\edef\tmpb{\replstring \tmpb {+}{_addcolor} \replstring \tmpb {-}{_addcolor-} \replstring \tmpb {+ }{+} \replstring \tmpb {- }{-} \replstring \tmpb {+ }{+} \replstring \tmpb {- }{-} \edef\tmpb{\addcolor{100.}{\#6}}}
\edef\tmpb{\setcmykcolor \{(\#1 \#2 \#3 \#4 \#5)\}}
\edef\tmpb{\setrgbcolor \{(\#1 \#2 \#3 \#4 \#5)\}}
\edef\tmpb{\setrgbcolor \{(\#1 \#2 \#3 \#4 \#5)\}}
\edef\tmpb{\directlua{color_C=\max{(color_C \#1 \#2 \#3 \#4 \#5)}}
\edef\tmpb{\color_Y=\max{(color_Y \#1 \#2 \#3 \#4 \#5)}}
\edef\tmpb{\color_K=\max{(color_K \#1 \#2 \#3 \#4 \#5)}}
\edef\tmpb{\edef\addcolor{\{(\#1 \#2 \#3 \#4 \#5)\}}
\edef\tmpb{\edef\addcolor\{(\#1 \#2 \#3 \#4 \#5)\}}}
\endgroup}
\end{verbatim}

119
2.21 The .ref file

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

\Xrefversion{\ref-version}\n\Xpage{\pagename}{\pagename}\n\Xtoc{\level}{\type}{\text}{\title}\n\Xlabel{\label}{\text}\n...\n\Xpage{\pagename}{\pagename}\n\Xlabel{\label}{\text}\n...
The \inputref macro is executed in \everyjob. It reads the \jobname\.ref file, if it exists. After the file is read then it is removed and opened for writing.

If the \jobname.ref file does not exist, then it is not created by default. This means that if you process a document without any forward references then no \jobname\.ref file is created (it would be unusable). The \wref macro is a dummy in that case.

If a macro needs to create and use the \jobname.ref file, then such macro must first use \openref. It creates the file and redefines \wref{\macro}{\data} so that it saves the line \macro{\data} to the \jobname.ref file using the asynchronous write primitive. Finally, \openref destroys itself, because we don’t need to open the file again.

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

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

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

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

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

2.22 References

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

\_Xpage \{(gpageno)\}\{(pageno)\} 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. It is set but unused in OpTeX versions 1.04+. You can add the report, for example:

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

\_Xlabel \{\langle label\rangle\}\{\langle text\rangle\} saves the \langle text\rangle to \_lab:\langle label\rangle and saves \{\langle gpageno\rangle\}\{\langle pageno\rangle\} to \_pgref:\langle label\rangle.
\label{[\textbf{label}]} saves the declared label to \texttt{\_lastlabel} and \texttt{\wlabel{\textbf{\langle text\rangle}}} uses the \texttt{\_lastlabel} and activates \texttt{\_wref\_Xlabel{\textbf{\langle label\rangle}}} {\textbf{\langle text\rangle}} uses the \texttt{\_lastlabel} and \texttt{\_wref\_Xlabel{\textbf{\langle label\rangle}}} {\textbf{\langle text\rangle}}.

\begin{verbatim}
\def\label{\ifempty{\@}{\global\let\_lastlabel=\undefined}{\iftrue\xdef\_lastlabel{\@}\fi}}
\def\wref#1#2{\ifx\_lastlabel\undefined\else\dest[ref:\_lastlabel]\fi}
\def\reftext#1#2{\ifatin#2@false#2\else\reftextA{#1}#2\fi}
\def\reftextA #1#2#3 {#2#1#3}
\def\isatin #1#2 {\ifx\fin#2\fin}
\def\printlabel#1{\vbox to \zo{\vss \llap{\labelfont[\@]}} \kern 1.7ex}
\fontdef \labelfont{\setfontsize{at10pt}{\setfontcolor{blue}{\tt}}}
\def\showlabels{\def\printlabel{\edef\printlabel#1{}}}
\end{verbatim}

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

\pgref{[\textbf{\langle given-text\rangle}]} prints \textbf{\langle given-text\rangle} where \textbf{@} is replaced by \textbf{\langle pageno\rangle}. Data in the format \textbf{\langle pageno\rangle} \textbf{\langle pageno\rangle} are read from \texttt{\pgref{\textbf{\langle label\rangle}}} by \texttt{\pgrefB{\textbf{\langle pageno\rangle}}{\textbf{\langle pageno\rangle}}{\textbf{\langle given-text\rangle}}}.

\texttt{\_lastreflabel} keeps the value of the last label read by \texttt{\ref} or \texttt{\pgref}. You can use it for example by defining a macro \texttt{\pg} by \texttt{\def\pg\pgref{\textbf{\langle lastreflabel\rangle}}} and then you need not repeat the same label in typical situations and you can write for instance: \texttt{see section \ref{\textbf{\langle label\rangle}}} at page \texttt{\pg}.

\begin{verbatim}
\def\ref{\@}{\reftextA{\@}{\textbf{\langle given-text\rangle}}}\\
\def\pgref{\@}{\reftextA{\@}{\textbf{\langle pageno\rangle}}}
\end{verbatim}

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

Default \texttt{\printlabel} is empty macro (labels are not printed). The \texttt{\showlabels} redefines it as box with zero dimensions and with left lapped \textbf{\langle label\rangle} in blue \texttt{\setfontsize{at10pt}{\setfontcolor{blue}{\tt}}} font shifted up by 1.7ex.

\begin{verbatim}
\def\printlabel{\edef\printlabel#1{}}
\def\showlabels{\def\printlabel{\vbox to \zo{\vss \llap{\labelfont[#1]}}} \kern 1.7ex}}
\fontdef \labelfont{\setfontsize{at10pt}{\setfontcolor{blue}{\tt}}}
\end{verbatim}
2.23 Hyperlinks

There are six types of internal links and one type of external link used in \texttt{OpTeX}. They are used in the format \texttt{(type):(spec)}.

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

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

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

The color for active links can be declared by \texttt{\def\_(type):linkcolor}, the border around link can be declared by \texttt{\def\_(type):border}. These macros are not declared by default, so color for active links are given only by \texttt{\hyperlinks} macro and borders are invisible. For example \texttt{\def\tocborder{\Red}} means that links from table of contents are in red. Another example \texttt{\def\tocborder{1 0 0}} causes red frames in TOC (not visible, only visible in PDF viewers).

\begin{verbatim}
\def\dest{(type):(spec)} \def\destactive{(type):(spec)}
\end{verbatim}

\texttt{\dest{(type):(spec)}} creates a destination of internal links. The destination is declared in the format \texttt{(type):(spec)}. If the \texttt{\hyperlinks} command in not used, then \texttt{\dest} does nothing else it is set to \texttt{\destactive}. The \texttt{\destactive} is implemented by \texttt{\pdfdest} primitive. It creates a box in which the destination is shifted by \texttt{\destactive}. 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.

\begin{verbatim}
\def\destactive{1.4em}
\def\destactive[#1:#2]{\if$#2$\else\ifvmode
  \vbox to \zo{\kern-\destactive \pdfdest name(#1) xyz \vss}}\fi\fi
\def\destbox[#1:#2]{\else \destbox[#1:#2]}
\def\destbox[#1]{\vbox to \zo{\kern-\destactive \pdfdest name(#1) xyz \vss}}
\def\dest[#1]{1}
\public \dest
\end{verbatim}

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

The \texttt{\linkdims} macro declares the dimensions of link area. A specific action can be defined for each link \texttt{(type)} by the macro \texttt{\action{(spec)}}. \texttt{OpTeX} defines only \texttt{\urlaction{(url)}}. The default link action (when \texttt{(type):action} is not defined) is \texttt{goto name{(type):(spec)}} (an internal link). It is declared in the \texttt{\linkactions{(type):(spec)}} macro. The \texttt{\pdfstartlink} primitive uses \texttt{attr{\pdfborder{(type)}}}. The \texttt{\pdfborder{(type)}} macro expands to \texttt{\if$#2$$/\Border[0 0 .6]} if the \texttt{(type):border} macro (i.e. \texttt{\refborder}, \texttt{\citeborder}, \texttt{\tocborder}, \texttt{\pgborder}, \texttt{\urlborder}, \texttt{\fntborder} or \texttt{\fnfborder}) is defined.

\begin{verbatim}
\protected\def\linkactive#1#2#3#4{(\quitvmode
  \pdfstartlink \linkdims attr{\pdfborder(#1)} \linkactions(#1)(#2) \relax
  \localcolor{\trycs{\linkcolor(#3)#4}} \pdfendlink
})
\protected\def\xlink#1#2#3#4{(\quitvmode
  \localcolor{\trycs{\linkcolor(#3)#4}} \pdfendlink
})
\end{verbatim}
The text version of the normal characters, say \texttt{The URL can be broken at any place using these default values. If you want to disable breaking between normal characters, penalty 100 is inserted after special characters, \texttt{nobreak} before special characters.}

\begin{verbatim}
\def\link{\{(type)\{(color)\{(text)\}} creates a link. It is kept here for backward compatibility and it is equivalent to \texttt{\_url\{(type)\{(color)\{(text)\}}} \texttt{. If \{(type)\} action is not defined then \texttt{\_link} creates internal link do the \texttt{\_dest\{(type)\{(spec)\}} \texttt{. You can have more links with the same \{(type)\} but only one \texttt{\_dest} in the document.}

\def\link{\{(type)\{(spec)\}} \texttt{is equivalent to \texttt{\_link} but the \{(color)\} is used from \texttt{\_hyperlinks} declaration (or it is overwritten by \texttt{\_def\{(type)\}linkcolor).}

\def\url{\{(url)\}} \texttt{creates external link. The \texttt{\_url} is detokenized with \texttt{\_escapechar=-1} before it is used, so \texttt{\%\#} etc. can be used in the \texttt{\url}.}

\public \_link \url \link ;

\hyperlinks \texttt{(link color)\url (link color) activates \texttt{\_dest}, \texttt{\_link}, so that they create links. Not setting colors \texttt{\_hyperlinks\{(\)}} is also supported.}

\url{\{(url)\}} \texttt{does approximately the same as \texttt{\_url\{(url)\}}} \texttt{, but more work is done before the \texttt{\url} is processed. The link-version of \texttt{\url} is saved to \texttt{\_tmpa} and the printed version in \texttt{\_tmpb}. The printed version is processed in four steps: 1. \texttt{\_url} \texttt{is replaced by \_urla} \texttt{, \_ur1b and \_urlc. The step 4 do following: The \texttt{\_urlskip} is inserted between each pair of "normal characters", i.e. characters not declared by \texttt{\_urlgskip} \texttt{. The special characters declared by \texttt{\_def\url\{(character)\}} are replaced by the body of their corresponding macro. The \texttt{\_urlskip}, \texttt{\_urlskipg}, \texttt{\_urlgskip} are typical skips used for special characters, their meaning is documented in the code below. You can change them. Default values: penalty 9990 is inserted between each pair of normal characters, penalty 100 is inserted after special characters, nobreak before special characters. The URL can be broken at any place using these default values. If you want to disable breaking between normal characters, say \texttt{\let\_urlskip\=\nobreak.}

The text version of the \texttt{\url} is printed in \texttt{\_urlfont.}
\end{verbatim}
2.24 Making table of contents

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

\Xtoc{(level)}{(type)}{(number)}{(a-title)}{(title)} (in .ref file) reads given data and appends them to the \_toclist as \_tocline{(level)}{(type)}{(number)}{(a-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
- (a-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 (title) parameter by \scantoeol from .ref file because the (title) can include something like `{`.

\maketoc{
\_toclist()
\_newifi \_ifischap \_ischapfalse
\_def\_Xtoc#1#2#3#4{\_ifnum#1=0 \_ischaptrue\_fi
\_addto\_toclist{(\_scantoeol\_XtocA)}
\_def\_XtocA#1{\_addto\_toclist{\_ea\_addto\_ea\_toclist\_ea\_currpage}}
\_tocline{(level)}{(type)}{(number)}{(a-title)}{(title)}{(gpageno)}{(pageno)} prints the record to the table of contents. It opens group, reduces \leftskip, \rightskip, runs the \everytocline (user can customise the design of TOC here) and runs \tocl{(level)} {(number)}{(title)}{(pageno)} macro. This macro starts with vertical mode, inserts one record with given (level) and it should end by \tocpar which returns to horizontal mode. The \tocpar appends \nobreak \hskip-2\_indent\null \par. This causes that the last line of the record is shifted outside the margin given by \rightskip. A typical record (with long (title)) looks like this:

| \llap{(number)} text text text text text text text text text text text text text text text text text text text text text .................... (pageno) |

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

The auxiliary macros are:

- \_llaptoclink\{(text)\} does _noindent_\_llap\{(linked text)\}.
- \_tocdotfill creates dots in the TOC.
- \_nofirst\macro applies the \macro only if we don’t print the first record of the TOC.
- \_tocpar finalizes one TOC record whith \_hfill\_pgn\{(pageno)\} as link to real \(gpage\) saved in \#6 of \_tocline. This is temporarily defined in the \_tocline.

If you want a special formating of TOC with adding more special lines (no generated as titles from \_chap, \_sec, \_secc), you can define \_addtotoc\{(level)\}\{(type)\}\{(number)\}\{(o-title)\}\{(title)\} macro:

\_addtotoc\{(level)\}\{(type)\}\{(number)\}\{(o-title)\}\{(title)\}

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

\_addtotoc\{(level)\}\{(type)\}\{(number)\}\{(o-title)\}\{(title)\}

Now, users can add a blue line into TOC by

\_addtotoc\{(blue-line)\}\{(relax)\}\{(blue text to be added in the TOC)\}

anywhere in the document. Note that \_relax in the fourth parameter means that outline will be not generated. And second parameter \_blue-line is only a comment (unused in macros).
\regmacro appends its parameters to _regtoc, _regmark and _regoul. These token lists are used in _maketoc, _begoutput and _pdfunidef.

\def _newtoks _regtoc _newtoks _regmark _newtoks _regoul
\toksapp _regtoc(#1)\toksapp _regmark(#2)\toksapp _regoul(#3) %
\def _regmacro #1#2#3{\toksapp _regtoc{#1}\toksapp _regmark{#2}\toksapp _regoul{#3} %
\def _newtoks _regtoc _newtoks _regmark _newtoks _regoul
\public _maketoc _regmacro ;

2.25 PDF outlines

2.25.1 Nesting PDF outlines

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

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

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

\advance \count2 by 1
% increases the mapping pointer of the type _ol:_count0:_count1:_count2 of this section
\advance _ol:_count0:_count1:_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.
2.25.2 Strings in PDF outlines

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

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

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

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

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

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

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

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

The \_prepinverb(macro)(separator)\{\text\} e.g. \_prepinverb\tmpb{aaa \bbb cccc \dd \ee} does \def\tmpb{\su \bbb\su\{ccc \dd\su\{ee\}} where \su is \scantextoken\unexpanded. It means that in-line verbatim are not argument of \scantextoken. First \edef\tmpb tokenizes again the \text but not the parts which were in the the in-line verbatim.

The \regmacro is used in order to set the values of macros \em, \rm, \bf, \it, \bi, \tt, \~/ and - to values usable in PDF outlines.
2.26 Chapters, sections, subsections

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

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

You can re-define \_printchap, \_printsec or \_printsecc macros if another design of section titles is needed. These macros get the ⟨title⟩ text in its parameter. The common recommendations for these macros are:

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

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

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

- You can use \_firstnoindent in order to the first paragraph after the title is not indented.
The `_sectionlevel` is the level of the printed section:

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

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

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

The `_the*` macros `\_thechapnum`, `\_thesecnum`, `\_theseccnum`, `\_thetnum`, `\_thefnum` and `\_thednum` include the format of numbers used when the object is printing. If chapter is never used in the document then `\chapnum=0` and `\othechapnum` expands to empty. Sections have numbers ⟨num⟩ and subsections ⟨num⟩. On the other hand, if chapter is used in the document then `\chapnum>0` and sections have numbers ⟨num⟩.⟨num⟩ and subsections have numbers ⟨num⟩.⟨num⟩.⟨num⟩.

The `\notoc` and `\nonum` prefixes are implemented by internal `\_ifnotoc` and `\_ifnonum`. 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 \[⟨label⟩\]. The `\chap`, `\sec`, and `\secc` macros read its parameter using `\scantoeol`. This causes that they cannot be used inside other macros. Use `\inchap`, `\insec`, and `\insecc` macros directly in such case.

The `\printrefnum[⟨pre⟩@⟨post⟩]` macro is used in `\print*` macros.

Note that the ⟨tite-text⟩ is `\detokenize`d before `\wref`, so the problem of “fragile macros” from old \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.

The `\thisoutline{⟨text⟩}` saves text to the `\theoutline` macro. `\printrefnum` uses it and removes it.
\_nbpar sets \interlinepenalty value. \_nl is "new line" in the text (or titles), but space in toc or headlines or outlines.

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

The \_mark (for running heads) is used in \_printsection only. We suppose that chapters will be printed after \vfil\break, so users can implement chapter titles for running headers directly by macros, no \_mark mechanism is needed. But sections need \_marks. And they can be mixed with chapter’s running heads, of course.

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

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

The \secl{⟨number⟩} {⟨title-text⟩}⟨eol⟩ should be used for various levels of sections (for example, when converting from Markdown to OpTeX). \secl1 is \chap, \secl2 is \sec, \secl3 is \secc and all more levels (for ⟨number⟩> 3) are printed by the common \_seclp macro. It declares only a simple design. If there is a requirement to use such more levels then the book designer can define something different here.
The \caption/{letter} increases \langle letter\rangle num counter, edefines \_thecapnum as \_the(letter)num and defines \_thecaptitle as language-dependent word using \_mtext, declares default format by \_captionformat/{letter}) and runs the \_everycaption(letter) tokens register. The two groups opened by \caption are finalized by first \_par from an empty line or from \vskip, \cskip or from \endinsert. If a } occurs first then \_par from \aftergroup is processed. The \_printcaption(letter) is called, it starts with printing of the caption.

The \_cskip macro inserts nonbreakable vertical space between the caption and the object.

\begin{verbatim}
347 \def \_captionformat#1{\nospacetimeafter \capa}
348 \optdef \_capa{}{\trylabel \_incaption}
349 \def \_incaption {\bgroupanother\par}
350 \_ifcsname \_\tmpa num\_endcsname \_ea\_incr \_csname \_\tmpa num\_endcsname\x
351 \_edef \_thecapnum {\_csname \_the\_\tmpa num\_endcsname}\
352 \_edef \_thecaptitle{\_mtext\_\tmpa}\
353 \_ea\_captionformat\_\tmpa\
354 \_ea\_the \_csname \everycaption\_\tmpa\_endcsname\
355 \_def \_par{\ifhmode\_nbpar\_egroup\_egroup\_fi}\
356 \_bgroup \_aftergroup \_par\
357 \_cs{_printcaption\_\tmpa}\
358 }
359 \def \_cskip {\_par\_nobreak\_medskip} % space between caption and the object
360 \public \caption \cskip ;
\end{verbatim}

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

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

The \_printcaptiont (or f) is processed inside group and the \_par can be run after this group. If you want to re-define formating parameters for \_par, do this in the macro \_captionformat. The \_captionsep inserts a separator between auto-generated caption number and the following caption text. Default separator is \_enspace but if the caption text starts with dot or colon, then the space is not inserted. A user can write \caption/t: My table and "T a ble 1.1: My table" is printed. You can re-define the \_captionsep macro if you want to use another separator.

\begin{verbatim}
385 \_def \_printcaption {\noindent \wlabel\_thecapnum {\bf\_thecaptitle~\_thecapnum}\
386 \_futurelet\_next\captionsep\
387 }
388 \def \_captionsep{\ifx\_next.\_bfnext \else\ifx\_next:\_bfnext\_bf\fi\fi}
389 \def \_bfnext#1{{\bf#1}}
390 \_let \_printcaptionf = \_printcaption % caption of figures = caption of tables
\end{verbatim}

If you want to declare a new type of \caption with independent counter, you can use following lines, where \_printcaptiona for Algorithms are declared:

\begin{verbatim}
\let\_printcaptiona = \_printcaptionf \let\_everycaptiona = \_everycaptionf
\newcount\_anum \addto\secx {\_anum=0 }
\_def\_theanum {\_othe\chapnum.\_the\secnum.\_the\_anum}
\_sdef {_mt:a:en}{Algorithm} \_sdef {_mt:a:cs}{Algoritmus} \% + your language...
\end{verbatim}

The format of the \_caption text is given by the \_captionformat/{caption-letter} macro. The default format for t and f is a paragraph in block narrower by \_iindent and with the last line is centered. This setting is done by the \_narrowlastlinecentered macro.
\texttt{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}).

2.27 Lists, items

\texttt{aboveliskip} is used above the list of items,\texttt{belowliskip} is used below the list of items,\texttt{setlistskip} sets the skip dependent on the current level of items,\texttt{listskipab} is \texttt{\listskipamount} or \texttt{\olistskipamount}.

The \texttt{printnumberedpar} \texttt{\theXnum} \texttt{\{name\}} 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:

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

\theorem Let $M$ be... \endtheorem
\end{verbatim}
Each level of nested lists is indented by the new \iindent from left. The default item mark is \printitem.

The \begitems runs \abovevskip 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).

\begitems \enditems \itemnum;

\novspaces sets \listskipab and \itemskipamount to 0pt. Moreover, it deactivates \setlistskip (for inner lists).

Various item marks are saved in \item:⟨letter⟩ macros. You can re-define them or define more such macros. The \style \item:⟨letter⟩ does \printitem={\item:⟨letter⟩} when it is used and finally, \startitem alias * uses \printitem.

\begblock macro selects fonts from footnotes \fnset and opens new indentation in a group. \endblock closes the group. This is implemented as an counterpart of Markdown’s Blockquotes. 

The \begblock macro selects fonts from footnotes \fnset and opens new indentation in a group. \endblock closes the group. This is implemented as an counterpart of Markdown’s Blockquotes. 

The \begblock macro selects fonts from footnotes \fnset and opens new indentation in a group. \endblock closes the group. This is implemented as an counterpart of Markdown’s Blockquotes. 

Redefine
these macros if you want to declare different design. The \texttt{OPTeX trick 0031} shows how to create blocks
with grey background splittable to more pages.

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

\section{Verbatim, listings}

\subsection{Inline and “display” verbatim}

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

When \texttt{\code} occurs in PDF outlines then it does the same as \texttt{\detokenize}. The macro for preparing outlines sets \texttt{\_escapechar} to \texttt{-1} and uses \texttt{\_regoul} token list before \texttt{\edef}.

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

\begin{verbatim}
\def\_code#1{\unexpanded\_{\csname \_code \_endcsname}{#1}}
\protected\sdef{_code }#1{{\escapechar=-1 \ttfont \the\everyintt \relax
\ea\printinverbatim\ea{\detokenize{#1}}}}
\def\printinverbatim#1{\leavevmode\hbox{#1}}
\regmacro {}{}{\let\code=\detokenize \let\_code=\detokenize}
\public \code ;
\end{verbatim}

The \texttt{\_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 \texttt{\_catcodetable\_verbatimcatcodes}. After the group is finished then original catcode table is restored.

\begin{verbatim}
\newcatcodetable \verbatimcatcodes
\def\_setverb{
  \begingroup
  \_def\do##1{\catcode`##1=12 }
  \dospecials
  \savecatcodetable\verbatimcatcodes % all characters are normal
  \_endgroup
  \_setverb
  \_def\_setverb{
    \_catcodetable\verbatimcatcodes %

  \verbchar\langle char⟩ saves original catcode of previously declared \langle char⟩ (if such character was declared)
using \verbchar\_savedttchar and \verbchar\_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 \langle text⟩ by \_printinverbatim and closes group. Suppose that \verbchar\_“\texttt{”} is
used. Then the following work is schematically done:

\begin{verbatim}
\def "{\begingroup \_setverb \_readverb}
\def \_readverb #1"{\_printinverbatim#1\_endgroup}
\end{verbatim}

\end{verbatim}
Note that the second occurrence of " is not active because _setverb deactivates it.

\begtt is defined only as public. We don’t need a private \begtt variant. This macro opens a group and sets _ as an active character (temporary). This will allow it to be used as the comment character at the same line after \begtt. Then \begtt is run. It is defined by \eoldef, so users can put a parameter at the same line where \begtt is. This # parameter is used after \everytt parameters settings, so users can change them locally.

The \begtt macro does _verb and another preprocessing, sets \endlinechar to \^J and reads the following text in verbatim mode until \endtt occurs. This scanning is done by \startverb macro which is defined as:

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

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

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

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

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

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

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

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

The \printverbline{(line)} expects that it starts in vertical mode and it must do \par to return the vertical mode. The \printverbline is used here: it does nothing when \ttline<0 else it prints the line number using \llap.
\puttpenalty puts \ttpenalty before second and next lines, but not before first line in each \begtt...\endtt environment.

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

\verbinput uses a file read previously or opens the given file. Then it runs the parameter scanning by \_viscanparameter and \_viscanminus. Finally the \_doverbinput is run. At the beginning of \_doverbinput, we have \_viline= number of lines already read using previous \verbinput, \_vinolines= the number of lines we need to skip and \_vidolines= the number of lines we need to print. A similar preparation is done as in \begtt after the group is opened. Then we skip \_vinolines lines in a loop a and we read \_vidolines lines. The read data is accumulated into \_tmpb macro. The next steps are equal to the steps done in \_startverb macro: data are processed via \_prepareverbdata and printed via \_printverb loop.
\_savemathsb, \_Restoremathsb pair is used in \begtt...\endtt or in \verbinput to temporary suppress the \mathsbon because we don’t need to print \verbtext in verbatim mode if \verbtext is really written. The \_Restoremathsb is defined locally as \_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 ⟨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
There is one limitation when \TeX{} interprets the comments declared by \commentchars{}. Each block of comments is accumulated to one line and then it is re-interpreted by \TeX{}. So, the ends of lines in the comments block are lost. You cannot use macros which need to scan end of lines, for example \begtt...\endtt inside the comments. The character \% is ignored in comments but you can use \% alone for de-activating \_endpar from empty comment lines.

Implementation: The \commentchars{first}{second} redefines the \testcommentchars{} in order to it removes the following \_iftrue and returns \_iftrue or \_iffalse depending on the fact that the comment characters are or aren’t present at the beginning of tested line. If it is true (\ifnum expands to \ifnum 10>0) then the rest of the line is added to the \vcomments{} macro.

The \_hicomments{} is \relax by default but it is re-defined by \commentchars{} in order to keep no-colored comments if we need to use feature from \commentchars{}.

The accumulated comments are printed whenever the non-comment line occurs. This is done by \printcomments{} macro. You can re-define it, but the main idea must be kept: it is printed in the group, \reloding \_rm initializes normal font, \catcodetable0 returns to normal catcode table used before \verbinput is started, and the text accumulated in \vcomments{} must be printed by \scantextokens{} primitive.

The \visiblesp sets spaces as visible characters \␣. It redefines the \dsp, so it is useful for verbatim modes only.

The \dsp is equivalent to \␣ primitive. It is used in all verbatim environments: spaces are active and defined as \dsp here.

2.28.2 Listings with syntax highlighting

The user can write

\begtt \hisyntax{C} \\
... \\
\endtt

to colorize the code using C syntax. The user can also write \everytt={\hisyntax{C}} to have all verbatim listings colorized.

\hisyntax{(name)} reads the file \hisyntax-(name).opm where the colorization is declared. The parameter (name) is case insensitive and the file name must include it in lowercase letters. For example, the file \hisyntax-c.opm looks like this:
OpTeX provides hisyntax-\{c, lua, python, tex, html\}.opm files. You can take inspiration from these files and declare more languages.

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

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

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

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

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

The pre-processing phase does:
The macros `\replfromto{<from>}{<to>}{<replacement>}` and `\replthis{<pattern>}{<replacement>}` manipulate the verbatim text that is already stored in the `\tmpb` macro. `\replfromto{<from>}{<to>}{<replacement>}` finds the first occurrence of `<from>` and the first occurrence of `<to>` following it. The `<text>` between them is packed into `\#1` and available to `<replacement>` which ultimately replaces `<text>.

`\replfromto` continues by finding next `<from>`, then, next `<to>` repeatedly over the whole verbatim text. If the verbatim text ends with opening `<from>` but has no closing `<to>`, then `<to>` is appended to the verbatim text automatically and the last part of the verbatim text is replaced too.

The first two parameters are expanded before use of `\replfromto`. You can use `\csstring` or something else here.

The `\replthis{<pattern>}{<replacement>}` replaces each `<pattern>` by `<replacement>`. Both parameters of `\repthis` are expanded first.

The macros are working only in a group when processing the verbatim text.

Special macros are working only in a group when processing the verbatim text.

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

- `\n` represents nothing but it should be used as a boundary of words as mentioned above.
- `\_t` represents a tabulator. It is prepared as `\n\n\n\n` because it can be at the boundary word boundary.
- `\x {<text>}` can be used as replacing text. Consider the example

```
\replfromto{/*}{*/}{\x C{/*#1*/}}
```

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

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

```
\x C{(text1)~^J(text2)~^J(text3)}
```

is replaced by

```
\x C{(text1)~^J\x C{(text2)~^J\x C{(text3)}}
```

The macro `\x {<text>}` is expanded to `\x: {<text>}` and if `\hicolor` `\{color\}` is declared then `\x: {<text>}` expands to `{<color>text}`. So, required color is activated for each line separately (e.g. for C comments spanning multiple lines).

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

The macros `\replfromto` and `\repthis` manipulate the verbatim text that is already stored in the `\tmpb` macro.

```
\_def\replfromto #1##2\{\_edef\tmpa{#1}{#2}\}\_ea\replfromtoE\_tmpa
```

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

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

```
\x C{(text1)~^J(text2)~^J(text3)}
```

is replaced by

```
\x C{(text1)~^J\x C{(text2)~^J\x C{(text3)}}
```

The macros `\replfromto` and `\repthis` manipulate the verbatim text that is already stored in the `\tmpb` macro.

```
\_def\replfromto #1##2\{\_edef\tmpa{#1}{#2}\}\_ea\replfromtoE\_tmpa
```

The macros `\replfromto` and `\repthis` manipulate the verbatim text that is already stored in the `\tmpb` macro.

```
\_def\replfromto #1##2\{\_edef\tmpa{#1}{#2}\}\_ea\replfromtoE\_tmpa
```

The macros `\replfromto` and `\repthis` manipulate the verbatim text that is already stored in the `\tmpb` macro.

```
\_def\replfromto #1##2\{\_edef\tmpa{#1}{#2}\}\_ea\replfromtoE\_tmpa
```

The macros `\replfromto` and `\repthis` manipulate the verbatim text that is already stored in the `\tmpb` macro.

```
\_def\replfromto #1##2\{\_edef\tmpa{#1}{#2}\}\_ea\replfromtoE\_tmpa
```

The macros `\replfromto` and `\repthis` manipulate the verbatim text that is already stored in the `\tmpb` macro.

```
\_def\replfromto #1##2\{\_edef\tmpa{#1}{#2}\}\_ea\replfromtoE\_tmpa
```

The macros `\replfromto` and `\repthis` manipulate the verbatim text that is already stored in the `\tmpb` macro.
The patterns ⟨from⟩, ⟨to⟩ and ⟨pattern⟩ are not found when they are hidden in braces {...}. E.g.

\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 replaces occurrences of \x by \z in the post-processing phase. The construct \x ⟨letter⟩⟨text⟩ expands to \xscan ⟨letter⟩⟨text⟩ “–J”. If #3 is \fin then it signals that something wrong happens, the ⟨from⟩ was not terminated by legal ⟨to⟩ when \replfromto did work. We must to fix this by using the \xscanR macro.

The \hicolor ⟨letter⟩ ⟨color⟩ defines \z: ⟨letter⟩{⟨text⟩} as {⟨color⟩⟨text⟩}. It should be used in the context of \x ⟨letter⟩{⟨text⟩} macros.

\hisyntax⟨name⟩ re-defines default \prepareverbdata ⟨macro⟩{⟨verbtext⟩}, but in order to do it does more things: It saves {⟨verbtext⟩} to \tmpp, appends \n around spaces and “–J” characters in pre-processing phase, opens hisyntax-⟨name⟩.opm file if \hisyntax⟨name⟩ is not defined. Then \the\hisyntax⟨name⟩ is processed. Finally, the post-processing phase is realized by setting appropriate values to the \x and \y macros and doing \edef\tmpp{\tmpp}.

Aliases for languages can be declared like this. When \hisyntax{xml} is used then this is the same as \hisyntax{html}.

\sdef{\hialias:xml}{html}
\sdef{\hialias:json}{c}
2.29 Graphics

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

\newbox\mypic
\setbox\mypic = \hbox{\picw=3cm \inspic{\langle picture \rangle}}

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

This code downloads the picture data to the PDF output only once (when \setbox is processed). Each usage of \copy\mypic puts only a pointer to the picture data in the PDF.

If you want to copy the same picture in different sizes, then choose a “basic size” used in \setbox and all different sizes can be realized by the \transformbox{\langle transformation \rangle}{\copy\mypic}.

\inspic accepts old syntax \inspic{\langle filename \rangle \langle space \rangle} or new syntax \inspic{\langle filename \rangle}. So, we need to define two auxiliary macros \_inspic\textsubscript{A} and \_inspic\textsubscript{B}.

All \inspic macros are surrounded in \hbox in order user can write \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 \inkinspic is used. These macros are stored in the \_inkdefs tokens list and it is used locally in the group. The solution is borrowed from OPmac trick 0032.

\inkinspic accepts \langle filename \rangle \langle space \rangle or \langle filename \rangle. So, we need to define two auxiliary macros \_inkinspic\textsubscript{A} and \_inkinspic\textsubscript{B}.

Inkscape can save a picture to *.pdf file and labels for the picture to *.pdf\_tex file. The second file is in \LaTeX\ format (unfortunately) and it is intended to read immediately after *.pdf is included in order to place labels of this picture in the same font as the document is printed. We need to read this \LaTeX\ file by plain \TeX\ macros when \inkinspic is used. These macros are stored in the \_inkdefs tokens list and it is used locally in the group. The solution is borrowed from OPmac trick 0032.

\inkinspic accepts \langle filename \rangle \langle space \rangle or \langle filename \rangle. So, we need to define two auxiliary macros \_inkinspic\textsubscript{A} and \_inkinspic\textsubscript{B}.

Inkscape can save a picture to *.pdf file and labels for the picture to *.pdf\_tex file. The second file is in \LaTeX\ format (unfortunately) and it is intended to read immediately after *.pdf is included in order to place labels of this picture in the same font as the document is printed. We need to read this \LaTeX\ file by plain \TeX\ macros when \inkinspic is used. These macros are stored in the \_inkdefs tokens list and it is used locally in the group. The solution is borrowed from OPmac trick 0032.

Inkscape can save a picture to *.pdf file and labels for the picture to *.pdf\_tex file. The second file is in \LaTeX\ format (unfortunately) and it is intended to read immediately after *.pdf is included in order to place labels of this picture in the same font as the document is printed. We need to read this \LaTeX\ file by plain \TeX\ macros when \inkinspic is used. These macros are stored in the \_inkdefs tokens list and it is used locally in the group. The solution is borrowed from OPmac trick 0032.
Inkscape may generate \textbf{\textit{\textsc{TEXT}}}

\_def\textbf#1{\_begingroup\_let\_it\_bf #1\_endgroup}\%
\_def\textit#1{\_begingroup\_it #1\_endgroup}\
\_def\textsl#1{\_begingroup\_trycs{slant}{}\_it #1\_endgroup}\
}

\public \inkinspic ;
\pdfscale{\langle x-scale \rangle}{\langle y-scale \rangle} and \pdfrotate{\langle degrees \rangle} macros are implemented by \pdfsetmatrix primitive. We need to know the values of sin, cos function in the \pdfrotate. We use Lua code for this.

\_def\_pdfscale#1#2{\_pdfsetmatrix{#1 0 0 #2}}
\_def\_gonfunc#1#2{\_directlua{tex.print(string.format('\_pcent.4f',math.#1(3.14159265*(#2)/180)))}}
\_def\_sin{\_gonfunc{sin}}
\_def\_cos{\_gonfunc{cos}}
\_def\_pdfrotate#1{\_pdfsetmatrix{\_cos{#1} \_sin{#1} \_sin{(#1)-180} \_cos{#1}}}\\
\public \pdfscale \pdfrotate ;

The \transformbox{\langle transformation \rangle}{\langle text \rangle} is copied from OPmac trick 0046. The \rotbox{\langle degrees \rangle}{\langle text \rangle} is a combination of \rotsimple from OPmac trick 0101 and the \transformbox. Note, that \rotbox{-90} puts the rotated text to the height of the outer box (depth is zero) because code from \rotsimple is processed. But \rotbox{-90.0} puts the rotated text to the depth of the outer box (height is zero) because \transformbox is processed.

\def\_multiplyMxV #1 #2 #3 #4 {% matrix * (vvalX, vvalY)
\_tmpdim = #1\_vvalX \_advance\_tmpdim by #3\_vvalY
\_vvalY = #4\_vvalY \_advance\_vvalY by #2\_vvalX
\_vvalX = \_tmpdim}
\def\_multiplyMxM #1 #2 #3 #4 {% currmatrix := currmatrix * matrix
\_vvalX=#1pt \_vvalY=#2pt \_ea\_multiplyMxV \_currmatrix
\_edef\_tmpb{\_ea\_ignorept\_the\_vvalX\_space \_ea\_ignorept\_the\_vvalY}%
\_vvalX=#3pt \_vvalY=#4pt \_ea\_multiplyMxV \_currmatrix
\_edef\_currmatrix{\_tmpb\_space 112}
\def\_transformbox#1#2{\_hbox{\_setbox0=\_hbox{{#2}}\_dimendef\_vvalX 11\_dimendef\_vvalY 12 % we use these variables\_dimendef\_newHt 13 \_dimendef\_newDp 14 % only in this group\_dimendef\_newLt 15 \_dimendef\_newRt 16\_protected\_def \_pdfsetmatrix {\_pdfextension setmatrix} %
\_let\pdfsetmatrix=\_pdfsetmatrix}
\def\_setnewHtDp #1 #2 {\_vvalX=#1\_relax \_vvalY=#2\_relax \_ea\_multiplyMxV \_currmatrix\_ifdim\_vvalX<\_newLt \_newLt=\_vvalX \_fi \_ifdim\_vvalX>\_newRt \_newRt=\_vvalX \_fi \_ifdim\_vvalY>\_newHt \_newHt=\_vvalY \_fi \_ifdim\_vvalY>\_newDp \_newDp=\_vvalY \_fi}\_protected\_def \_pdfsetmatrix{\_pdfextension setmatrix}\_let\pdfsetmatrix=\_pdfsetmatrix
\def\_def\_rotsimple{\_pdfsetmatrix=\_pdfsetmatrix 1%}
\_def\_pdfsetmatrix[#1]\_edef\_tmpb{\_ea\_multiplyMxM{\_pdfsetmatrix\_tmpb\_unskip}}\_let\pdfsetmatrix=\_pdfsetmatrix #1%
\_setnewHtDp Opt \_ht0 \_setnewHtDp Opt \_dp0\_setnewHtDp \_wd0 \_setnewHtDp \_dp0
\protected\_def \_pdfsetmatrix{\_pdfextension setmatrix}\_let\pdfsetmatrix=\_pdfsetmatrix
\_def\_setnewHtDp #1 #2 {%\_vvalX=#1 \_relax \_vvalY=#2 \_relax \_ea\_multiplyMxM \_currmatrix\_ifdim\_vvalX<\_newLt \_newLt=\_vvalX \_fi \_ifdim\_vvalX>\_newRt \_newRt=\_vvalX \_fi\_ifdim\_vvalY<\_newHt \_newHt=\_vvalY \_fi \_ifdim\_vvalY>\_newDp \_newDp=\_vvalY \_fi\_ifdim\_vvalY>\_newHt \_newHt=\_vvalY \_fi \_ifdim\_vvalY>\_newDp \_newDp=\_vvalY \_fi}
\protected\_def \_rotbox#1#2{\_isequal{90}{#1}\_iftrue \_rotboxA{#1}{\_kern\_ht0 \_tmpdim=\_dp0}\_vfill{\_kern\_dp0}\_fi\_else \_isequal{-90}{#1}\_iftrue \_rotboxA{#1}{\_kern\_dp0 \_tmpdim=\_ht0}\_vfill{\_kern\_dp0}\_fi\_else \_transformbox{\_pdfrotate{#1}}{\_kern\_dp0}\_fi}
\_scantwodimens scans two objects with the syntactic rule \texttt{⟨dimen⟩} and returns \{\texttt{⟨number⟩}\}{\texttt{⟨number⟩}}

in sp unit.

\puttext \{\texttt{⟨right⟩}\}{\texttt{⟨up⟩}}{\texttt{⟨text⟩}} puts the \texttt{⟨text⟩} to desired place: From current point moves \{\texttt{⟨down⟩}\} and \{\texttt{⟨right⟩}\}, puts the \texttt{⟨text⟩} and returns back. The current point is unchanged after this macro ends.

\putpic \{\texttt{⟨right⟩}\}{\texttt{⟨up⟩}}{\texttt{⟨width⟩}}{\texttt{⟨height⟩}}{\texttt{⟨image-file⟩}} does \puttext with the image scaled to desired \texttt{⟨width⟩} and \texttt{⟨height⟩}. If \texttt{⟨width⟩} or \texttt{⟨height⟩} is zero, natural dimension is used. The \texttt{\nospec} is a shortcut to such a natural dimension.

\backgroundpic{\texttt{⟨image-file⟩}} puts the image to the background of each page. It is used in the \texttt{\slides} style, for example.

\circle{\texttt{⟨x⟩}}{\texttt{⟨y⟩}} creates an ellipse with \texttt{⟨x⟩} axis and \texttt{⟨y⟩} axis. The origin is in the center.

\oval{\texttt{⟨x⟩}}{\texttt{⟨y⟩}}{\texttt{⟨roundness⟩}} creates an oval with \texttt{⟨x⟩}, \texttt{⟨y⟩} size and with the given \texttt{⟨roundness⟩}. The real size is bigger by 2\texttt{⟨roundness⟩}. The origin is at the left bottom corner.

\mv{\texttt{⟨x⟩}}{\texttt{⟨y⟩}}{\texttt{⟨curve⟩}} moves current point to \texttt{⟨x⟩}, \texttt{⟨y⟩}, creates the \texttt{⟨curve⟩} and returns the current point back. All these macros are fully expandable and they can be used in the \texttt{\pdfliteral} argument.
The `\inoval{⟨text⟩}` is an example of `\oval` usage.
The `\incircle{⟨text⟩}` is an example of `\circle` usage.
The `\ratio, \lwidth, \fcolor, \lcolor, \shadow` and `\overlapmargins` are parameters, they can be set by user in optional brackets [ ... ]. For example `\fcolor=\Red` does `\let\fcolorvalue=\Red` and it means filling color.
The `\setflcolors` uses the `\setcolor` macro to separate filling (non-stroking) color and stroking color. The `\_coc` macro means “create oval or circle” and it expands to the stroking primitive `S` or filling primitive `f` or both `B`. Only boundary stroking is performed after `\fcolor=\relax`. You cannot combine `\fcolor=\relax` with `\_shadow=Y`.
Just before defining shadows, which require special graphics states, we define means for managing these graphics states and other PDF page resources (graphics states, patterns, shadings, etc.). Our mechanism, defined mostly in Lua (see 2.39.4, uses single dictionary for each PDF page resource type (extgstate, etc.) for all pages. \_pdfpageresources just points to it).

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

A shadow effect is implemented here. The shadow is equal to the silhouette of the given path in a gray-transparent color shifted by \_shadowmoveto vector and with blurred boundary. A waistline with the width 2\_shadowb around the boundary is blurred. The \_shadowlevels levels of transparent shapes is used for creating this effect. The \_shadowlevels+1/2 level is equal to the shifted given path.

The \_doshadow{⟨curve⟩} does the shadow effect.
A generic macro \texttt{\clipinpath(x) \{y\} \{curve\} \{text\}} declares a clipping path by the \texttt{\{curve\}} shifted by the \texttt{\{x\}}, \texttt{\{y\}}. The \texttt{\{text\}} is typeset when such clipping path is active. Dimensions are given by bp without the unit here. The macros \texttt{\clipinoval(x) \{y\} \{width\} \{height\} \{\text\}} and \texttt{\clipincircle(x) \{y\} \{width\} \{height\} \{\text\}} are defined here. These macros read normal TeX dimensions in their parameters.

2.30 The \texttt{\table} macro, tables and rules

2.30.1 The boundary declarator:

The \texttt{\{declaration\}} part of \texttt{\table\{\{declaration\}\}\{\text\}} includes column declarators (letters) and other material: the | or (\texttt{\langle cmd\rangle}). If the boundary declarator : is not used then the boundaries of columns are just before each column declarator with exception of the first one. For example, the declaration \texttt{\{c|c\(xx\)(yy)c\}} should be written more exactly using the boundary declarator : by \texttt{\{c| :c\(xx\)::(yy)c\}. But you can set these boundaries to other places using the boundary declarator : explicitly, for example \texttt{\{c|:c\(xx\):(yy)c\}}. The boundary declarator : can be used only once between each pair of column declarators.

Each table item has its group. The \texttt{\langle cmd\rangle} are parts of the given table item (depending on the boundary declarator position). If you want to apply a special setting for a given column, you can do this by \texttt{\langle setting\rangle} followed by column declarator. But if the column is not first, you must use \texttt{\langle setting\rangle}. Example. We have three centered columns, the second one have to be in bold font and the third one have to be in red: \texttt{\table\{c:\{bf\}c:\{Red\}c\}\{\text\}}
2.30.2 Usage of the \tabskip primitive

The value of \tabskip primitive is used between all columns of the table. It is glue-type, so it can be stretchable or shrinkable, see next section 2.30.3.

By default, \tabskip is 0 pt. It means that only \tabiteml, \tabitemr and ⟨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 \tabskip value declared before the \table macro (or in \everytable or in \thistable) is used between all columns in the table. This value is equal to all spaces between columns. But you can set each such space individually if you use ⟨\tabskip=value⟩ in the ⟨declaration⟩ immediately before boundary character. The boundary character represents the column pair for which the \tabskip has individual value. For example c(\tabskip=5pt):r gives \tabskip value between c and r columns. You need not use boundary character explicitly, so c(\tabskip=5pt)r gives the same result.

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

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

2.30.3 Tables to given width

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

- \table to⟨size⟩{⟨declaration⟩}{⟨data⟩} uses stretchability or shrinkability of all spaces between columns generated by \tabskip value and eventually by \tabskipl, \tabskipr values. See example below.

- \table pxto⟨size⟩{⟨declaration⟩}{⟨data⟩} expands the columns declared by p⟨⟨size⟩⟩, if the ⟨size⟩ is given by a virtual \tsize unit. See the example below.

Example of \table to⟨size⟩:

\thistable{\tabskip=0pt plus1fil minus1fil}
\table to\hsize {lr}{⟨data⟩}

This table has its width \hsize. The first column starts at the left boundary of this table and it is justified left (to the boundary). The second column ends at the right boundary of the table and it is justified right (to the boundary). The space between them is stretchable and shrinkable to reach the given width \hsize.

Example of \table pxto⟨size⟩ (means “paragraphs expanded to”):

\table pxto\hsize {l|c|p{\tsize}|}{\crl
aaa & Ddkas jd dsjds ds cgha sfgs dd fddzf dfhz xxz dkas jd dsjds ds cgha sfgs dd fddzf dfhz xxz. \crl
bb ddd ggg & Dajds ds cgha sfgs dd fddzf dfhz xxz dkas jd dsjds ds cgha sfgs dd fddzf dfhz xxz.
}

The first c column is variable width (it gets the width of the most wide item) and the resting space to given \hsize is filled by the p column.

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

\table pxto13cm {r p{3.5\tsize} p{2\tsize} p{\tsize} l}{⟨data⟩}

This gives the ratio of widths of individual paragraphs in the table 3.5:2:1.
2.30.4 \texttt{\eqbox}: boxes with equal width across the whole document

The \texttt{\eqbox \{\langle label\rangle\}\{\langle text\rangle\}} behaves like \texttt{\hbox\{\langle text\rangle\}} in the first run of \TeX. But the widths of all boxes with the same label are saved to \texttt{.ref} file and the maximum box width for each label is calculated at the beginning of the next \TeX run. Then \texttt{\eqbox \{\langle label\rangle\}\{\langle text\rangle\}} behaves like \texttt{\dimbox to \{\langle dim:label\rangle\}\{\langle text\rangle\}\hss}, where \texttt{\{\langle dim:label\rangle\}} is the maximum width of all boxes labeled by the same \texttt{\{\langle label\rangle\}}. The documentation of the \LaTeX{} package \texttt{eqparbox} includes more information and tips.

The \texttt{\eqboxsize \{\langle label\rangle\}\{\langle dimen\rangle\}} expands to \texttt{\dimbox to \{\langle dimen\rangle\}} if this value is known, else it expands to the given \texttt{\{\langle dimen\rangle\}}.

The optional parameter \texttt{r} or \texttt{l} can be written before \texttt{\{\langle label\rangle\}} (for example \texttt{\eqbox r\{label\}\{text\}}) if you want to put the text to the right or to the left side of the box width.

Try the following example and watch what happens after first \TeX run and after the second one.

\begin{verbatim}
def\leftitem#1{\par
\noindent \hangindent=\eqboxsize\{items\}\{2em\}\hangafter=1
\eqbox r\{items\}\{#1\}\ignorespaces}
def\leftitem{{\bf first}}\lorem{1}
def\leftitem{{\bf second one}}\lorem{2}
def\leftitem{{\bf final}}\lorem{3}
\end{verbatim}

2.30.5 Implementation of the \texttt{\table} macro and friends

The result of the \texttt{\table\{\langle declaration\rangle\}\{\langle 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{p nto\{size\}} to \texttt{\_tablew} and \texttt{\_tableW} sets the \texttt{\_tablew} macro. If \texttt{\_tmpdim} is used then \texttt{\_tablew} is empty and \texttt{\_tmpdim} includes given \texttt{\{size\}}. The \texttt{\_ifpxto} returns true in this case.

The \texttt{\table} continues by reading \texttt{\{\langle 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{\catcodetable} here.

The \texttt{\tablinespace} 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 declaration\rangle\}} macro scans the \texttt{\{\langle declaration\rangle\}} by \texttt{\_scantabdata\#1\_relax} and continues by processing \texttt{\{\langle data\rangle\}} by \texttt{\_tableB}. The trick \texttt{\_tmptoks={\langle\langle data\rangle\rangle}} \texttt{\edef\_tmpb{\_the\_tmptoks}} is used here in order to keep the hash marks in the \texttt{\{\langle data\rangle\}} unchanged.
The \_tableB saves \langle data\rangle to \_tmpb and does \_replstrings to prefix each macro \crl (etc.) by \_crcr. See \_tabreplstrings. The whole \_tableB macro is hidden in \{...\} in order to there may be \table in \table and we want to manipulate with & and \cr as with normal tokens in the \_tabreplstrings, not as the item delimiters of an outer \table.

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

\_tabskip=\_tabskipmid \_tabreplstrings \_tableB \_tableC

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

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

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

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

\_tablew replaces each \crl etc. to \crcr\crl. The reason is: we want to use macros that scan its parameter to a delimiter written in the right part of the table item declaration. The \crcr cannot be hidden in another macro in this case.

The \_tabdata macro converts \_table’s \langle declaration \rangle to \_table\langle converted declaration \rangle. The result is stored into \_tabdata tokens list. For example, the following result is generated when \langle declaration\rangle=|\crl|\_crl|
The second result in the \_ddlinedata macro is a template of one row of the table used by \crli macro.

```
150 \_def\_scantabdata#1{151 \_let\_next=\_scantabdata
152 \_ifx\_relax#1\_let\_next=\_relax
153 \_else\_ifx|#1\_addtabvrule
154 \_else\_ifx:#1\_def\_next{\_scantabdataE}%
155 \_else\_ifx:#1\_def\_next{\_scantabdataF}%
156 \_else\_isinlist{123456789}#1\_iftrue \_def\_next{\_scantabdataC#1}%
157 \_else \_ea\_ifx\_csname _tabdeclare#1\_endcsname \_relax
158 \_ea\_ifx\_csname _paramtabdeclare#1\_endcsname \_relax
159 \_opwarning{tab-declarator "#1" unknown, ignored}%
160 \_else\_def\_next{\_ea\_scantabdataB\_csname _paramtabdeclare#1\_endcsname} \_fi
161 \_else \_def\_next{\_ea\_scantabdataA\_csname _tabdeclare#1\_endcsname}\_fi \_next
162 }
163 \_def\_scantabdataA#1{164 \_addtabitem
165 \_ea\_addtabdata\_ea{#1\_tabstrutA \_tabskip\_tabskipmid\_relax}\_scantabdata
166 }
167 \_def\_scantabdataB#1#2{168 \_addtabitem
169 \_ea\_addtabdata\_ea{#1{#2}\_tabstrutA \_tabskip\_tabskipmid\_relax}\_scantabdata
170 }
171 \_def\_scantabdataC{172 \_def\_tmpb{}\_afterassignment\_scantabdataD \_tmpnum=}
173 \_def\_scantabdataD#1{174 \_loop \_ifnum\_tmpnum>0 \_advance\_tmpnum by-1 \_addto\_tmpb{#1}\_repeat
175 \_ea\_scantabdata\_tmpb}
176 \_def\_scantabdataE#1){177 \_addtabdata{#1}\_scantabdata}
178 \_def\_scantabdataF{179 \_addtabitem\_def\_addtabitem{\_let\_addtabitem=\_addtabitemx}\_scantabdata}
```

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

```
\_def\_showcolnum{\_ea\_def\_ea\_totcolnum\_ea{\_the\_colnum/\_the\_colnum}}
```

This example prints 1/3 2/3 3/3, because the value of the \_colnum is equal to the total number of columns before left part of the column declaration is processed.

```
\_newcount\_colnum % number of current column in the table
\_public \_colnum ;
```

This code converts || or | \_table ⟨declaration⟩ to the ⟨converted declaration⟩.

```
\_def\_addtabvrule{156 \_ifx\_tmpa\_vrule \_addtabdata{\_kern\_vvkern}\
157 \_ifnum\_colnum=0 \_addto\_dlinedata{\_vvitem}\_fi
158 \_advance\_colnum by1 \_let\_tmpa=\_relax
159 \_ifnum\_colnum=0 \_addto\_dlinedata{\_vvitem}\_fi
160 \_def\_addtabdata#1{\_dlinedata{#1}}}
```

The default “declaration letters” c, l, r and p are declared by setting \_tabdeclarec, \_tabdeclarel, \_tabdeclarer and \_paramtabdeclarep macros. In general, define \_def\_tabdeclare{letter} {...}
for a non-parametric letter and `\def\_paramtabdeclare\{letter\}{...}` for a letter with a parameter. The double hash `##` must be in the definition, it is replaced by a real table item data. You can declare more such “declaration letters” if you want.

Note, that the `##` with fills are in group. The reason can be explained by following example:

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

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

The `\_paramtabdeclare\{(data)\}` is invoked when `p\{(data)\}` declarator is used. First, it saves the `\hsize` value and then it runs `\_tablepar`. The `\_tablepar` macro behaves like `\_tableparbox` (which is `\vtop`) in normal cases. But there is a special case: if the first pass of `pxto` table is processed then `\hsize` is negative. We print nothing in this case, i.e. `\_tableparbox` is `\ignoreit` and we advance the `\_tsizesum`. The auxiliary macro `\_tsizelast` is used to do advancing only in the first row of the table. `\_tsizesum` and `\_tsizelast` are initialized in the `\_tableB` macro.

The `\_tableparB` initializes the paragraphs inside the table item and `\_tableparC` closes them. They are used in the `\_paramtabdeclare` macro. The first paragraph is no indented.

Users put optional spaces around the table item typically, i.e. they write `& text &` instead of `&text&`. The left space is ignored by the internal TeX algorithm but the right space must be removed by macros. This is a reason why we recommend to use `\_unsskip` after each `##` in your definition of “declaration letters”. This macro isn’t only the primitive `\unskip` because we allow usage of plain TeX `\hideskip` macro: `&\hideskip text\hideskip&`.

The `\_tableparD` is processed after the box is set: it checks if there is only one line and prints `\hbox to\hsize{\hfil\(this\line\)\hfil}` in this case. In other cases, the box2 is printed.
The family of \_cr* macros \_crl, \_crll, \_crli, \_crlli, \_crlp and \_tskip \(<\text{dimen}\) is implemented here. The \_zerotabrule is used to suppress the negative \_lineskip declared by \_tablinespace.

The \_mspan\(<\text{number}\)[\{\text{declaration}\}][\{\text{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\(<\text{number}\)[\{\text{text}\}]\) implementation is here. We need to lower the box by \((\text{number}-1)*\_ht\_dp\) of \_tabstrut) / 2.

The \#1 parameter must be a one-digit number. If you want to set more digits then use braces.

The parameters of primitive \_vrule and \_hrule keeps the rule “last wins”. If we re-define \_hrule to \_orihrule \_height\_ipt then each usage of redefined \_hrule uses 1pt height if this parameter isn’t
overwritten by another following \textit{height} parameter. This principle is used for settings another default rule thickness than 0.4pt by the macro \texttt{\rulewidth}.

The \texttt{\frame{⟨text⟩}} uses “\texttt{vbox in \texttt{vtop}}” trick in order to keep the baseline of the internal text at the same level as outer baseline. User can write \texttt{\frame{abcxyz}} in normal paragraph line, for example and gets the expected result: \texttt{abcxyz}. The internal margins are set by \texttt{\vvkern} and \texttt{\hhkern} parameters.

\texttt{\eqbox} and \texttt{\eqboxsize} are implemented here. The widths of all \texttt{\eqbox}es are saved to the \texttt{.ref} file in the format \texttt{\_Xeqbox{⟨label⟩}{⟨size⟩}}. The \texttt{.ref} file is read again and maximum box width for each \texttt{⟨label⟩} is saved to \texttt{\_eqb:⟨label⟩}.

2.31 Balanced multi-columns

\texttt{\betweencolumns} or \texttt{\leftofcolumns} or \texttt{\rightofcolumns} include a material printed between columns or left of all columns or right of all columns respectively. The \texttt{\betweencolumns} must include a stretchability or a material with exactly \texttt{\colsep} width. You can redefine these macros. For example the rule between columns can be reached by \texttt{\def \betweencolumns{\hss\vrule\hss}}.

The code used here is documented in detail in the “\TeXbook naruby”, pages 244–246, free available, \url{http://petr.olsak.net/tbn.html}, but in Czech. Roughly speaking, macros complete all material between \texttt{\beg multi} and \texttt{\end multi} into one \texttt{\vbox} 6. Then the macro measures the amount of free space at the current page using \texttt{\pagegoal} and \texttt{\pagtotal} and does \texttt{\vsplit} of \texttt{\vbox} 6 to columns with a height of such free space. This is done only if we have enough amount of material in \texttt{\vbox} 6 to fill the full page by columns. This is repeated in a loop until we have less amount of material in \texttt{\vbox} 6. Then we run \texttt{\balancecolumns} which balances the last part of the columns. Each part of printed material is distributed to the main vertical list as \texttt{\hbox{⟨columns⟩}} and we need not do any change in the output routine.
If you have paragraphs in `\begmulti` environment then you may say `\raggedright` inside this environment and you can re-assign `\widowpenalty` and `\clubpenalty` (they are set to 10000 in OpTeX).

\begin{verbatim}
\multiskip \def \Ncols{#1} \par
\begin{multicols}{#1}
\ifnum lastpenalty=10000 \vskip 4.5\baselineskip \penalty 9999 \vskip -4.5\baselineskip \fi
\multiskip \def \Ncols{#1} \vbadness 20000 \dimen 6 = \wd 6 \createcolumns \printcolumns
\divide \dimen 0 by \baselineskip \multiply \dimen 0 by \Ncols \global \advance \mullines by -\dimen 0 \egroup
\end{multicols}
\end{verbatim}

Splitting columns...

\begin{verbatim}
\def \makecolumns{\bgroup % full page, destination height: \dimen 1
\vbadness 20000 \dimen 6 = \wd 6 \createcolumns \printcolumns \dimen 0 = \dimen 1 \divide \dimen 0 by \baselineskip \multiply \dimen 0 by \Ncols \global \advance \mullines by -\dimen 0 \egroup
\splitpart{\% makecolumns % full page \vskip 0 pt plus 1fil minus \baselineskip \break \ifnum mullines<\tmpnum \dimen 0 = \ht 6 \else \dimen 0 = .8 \maxdimen \fi \divide \dimen 0 by \Ncols \relax \ifx balancecolumns \flushcolumns \advance \dimen 0 by - .5 \vsize \fi \ifvoid 6 \else \ifdim \dimen 0 > \dimen 2 \ea \ea \ea \splitpart \else \balancecolumns \fi \fi \divide \dimen 0 by \Ncols \relax \advance \dimen 2 by - \baselineskip \if\balancecolumns \else \splitpart \fi \fi \divide \dimen 0 by \baselineskip \if\balancecolumns \else \splitpart \fi \fi \if\balancecolumns \else \splitpart \fi % only balancing
\multiskip \egroup
\end{verbatim}

Final balancing of the columns.

\begin{verbatim}
\def \balancecolumns{\bgroup \setbox 7 = \copy 6 % destination height: \dimen 0
\ifdim \dimen 0 > \dimen 2 \ea \ea \splitpart \else \balancecolumns % last balancing
\fi \fi
\end{verbatim}

159
\_setcolsize(dimen) sets initial value \dimen1=\langle size\rangle which is used as height of columns at given page. The correction \splittopskip−\topskip is done if the columns start at the top of the page.

\_createcolumns prepares columns with given height \dimen1 side by side to the \box1.

\_printcolumns prints the columns prepared in \box1. The first \hbox{} moves typesetting point to the next baseline. Next negative skip ensures that the first line from splitted columns is at this position.

\_def\_setcolsize #1\{\_dimen1=#1\_relax
\_ifdim\_dimen1=\vsize\_advance\_dimen1 by \splittopskip \_advance\_dimen1 by−\topskip \_fi
}\_def\_createcolumns{\_setbox1=\hbox{\_leftofcolumns}\_tmpnum=0
\_loop \_ifnum\_Ncols>\_tmpnum \_advance\_tmpnum by1
\_setbox1=\hbox{\_unhbox1
\_ifvoid6 \_hbox to\_dimen6{\_hss}\_else \_vsplit6 to\_dimen1 \_fi
\_ifnum\_Ncols=\_tmpnum \_rightofcolumns \_else \_betweencolumns \_fi}\
\_repeat
\_def\_printcolumns{\hbox{}\_nobreak\_vskip−\splittopskip \_nointerlineskip
\_hbox to\_hsize{\_unhbox1}}
\_public "begmulti" "endmulti"

2.32 Citations, bibliography

2.32.1 Macros for citations and bibliography preloaded in the format cite-bib.opm

\_newcount\_bibnum % the bibitem counter
\_newtoks\_bibmark % the bibmark used if \_nonumcitations is set.
\_newcount\_lastcitenum \_lastcitenum=0 % for \_shortcitations
\_public \_bibnum \_bibmark ;

\_bibp expands to \_bibpart/. By default, \_bibpart is empty, so internal links are in the format cite:/(\langle number\rangle). If \_bibpart is set to \langle\_bibpart\rangle, then internal links are cite:\langle\_bibpart\rangle/\langle\_number\rangle.

\_def\_bibp\{\_the\_bibpart/\}
\_def\_nonumcitations\{\ contempt \_bibpart % sets \_bibpart to the empty string
\_def\_shortcitations\{\_bibpart % sets \_bibpart to something
\_public \_bibpart \_bibmark ;

\_bibnum expands to \_bibnum\_bibmark. \_bibmark is used when \_bibnum counts the bibliography items from one. The \_bibmark is used when \_nonumcitations is set.

\_def\cite\[\langle label\rangle,\langle label\rangle,...,\langle label\rangle\] % manages \langle labels\rangle using \_citeA and prints \{\langle bib-marks\rangle\} using \_printsavedcites.
\_def\nocite\[\langle label\rangle,\langle label\rangle,...,\langle label\rangle\] only manages \langle labels\rangle but prints nothing.
\_def\rcite\[\langle label\rangle,\langle label\rangle,...,\langle label\rangle\] behaves like \_cite but prints \{\langle bib-marks\rangle\} without brackets.
\_def\ecite\[\langle label\rangle\{\langle text\}\}\langle text\} behaves like \_rcite \[\langle label\rangle\] but prints \langle text\} instead \langle bib-mark\rangle. The \langle text\} is hyperlinked like \langle bib-marks\} when \_cite or \_rcite is used. The empty internal macro \_savedcites list to be printed. This list is set by \_citeA inside a group and it is used by \_printsavedcites in the same group. Each \_cite/\_rcite/\_ecite macro starts from empty list of \langle bib-marks\rangle because new group is opened.

\_def\_cite\[\#1\]{\_citeA\#1; \_printsavedcites}
\_def\_nocite\[\#1\]{\_citeA\#1; \_printsavedcites}
\_def\_rcite\[\#1\]{\_bgroup\_citeA\#1; \_ea\_eciteB\_savedcites;}
\_def\_ecite\[\#1\;\#2\;\#3\]{\_if\_if\#1\_relax \#3\_else \_link{\_cite:\_bibp\#1\(\#3\)}\_fi\_egroup}
\_def\_savedcites

\_public \_cite \_nocite \_rcite \_ecite ;


\citeB{\bibpart}{\label} processes one label from the list of labels given in the parameter of \cite, \nocite, \rcite or \ecite macros. It adds the \label to a global list \ctlst: {\bibpart} / which will be used by \usebib (it must know what \labels are used in the document to pick-up only relevant bib-entries from the database. Because we want to save space and to avoid duplications of \label in the \ctlst: {\bibpart} /, we distinguish four cases:

- (label) was not declared by \Xbib before and it is first such a \label in the document: Then \bib: {\bibpart} / {\label} is undefined and we save label using \adddcitelist, write warning on the terminal and define \bib: {\bibpart} / {\label} as empty.
- (label) was not declared by \Xbib before but it was used previously in the document: Then \bib: {\bibpart} / {\label} is empty and we do nothing (only data to \savedcites are saved).
- (label) was declared by \Xbib before and it is first such \label used in the document: Then \bib: {\bibpart} / {\label} includes \bibnn\{number\} \&_ if & \bibnn\{number\} \&_ . This is true when \bibnn\{number\} expands to empty. The \label is saved by \adddcitelist and \bib: {\bibpart} / {\label} is re-defined directly as \number.
- (label) was declared by \Xbib and it was used previously in the document. Then we do nothing (only data to \savedcites are saved).

The \citeA macro runs repeatedly over the whole list of \labels.
The ⟨bib-marks⟩ (in numeric or text form) are saved in \savedcites macro separated by commas. The \printsavedcites prints them by normal order or sorted if \sortcitations is specified or condensed if \shortcitations is specified.

The \sortcitations appends the dummy number 300000 and we suppose that normal numbers of bib-entries are less than this constant. This constant is removed after the sorting algorithm. The \shortcitations sets simply \lastcitenum=1. The macros for ⟨bib-marks⟩ printing follows (sorry, without detail documentation). They are documented in opmac-d.pdf (but only in Czech).
The \bib [\{label\}] or \bib [\{label\}] =\{\{bib-mark\}\} prints one bib-entry without reading any database. The bib-entry follows after this command. This command counts the used \bib from one by \bibnum counter and saves \Xbib\{\{bibpart\}\}\{\{label\}\}\{\{number\}\} into .ref file immediately using \wibib\{\{label\}\}\{\{number\}\}\{\{nonumber\}\}. This is the core of creation of mapping from \{labels\} to \{number\} and \{nonumber\}. \texttt{\_bibA} and \texttt{\_bibB} implement the scanner of the optional argument with the \texttt{\bibmark}. \texttt{\_bibgl} is \texttt{\relax} by default but \texttt{\slides} do \texttt{\let\_bibgl=\_global}. \texttt{\_dbib\{\{label\}\}} creates destination for hyperlinks.

The \texttt{\_printbib} prints the bib-entry itself. You can re-define it if you want a different design. The \texttt{\_pritbib} starts in horizontal mode after \texttt{\noindent} and after the eventual hyperlink destination is inserted. By default, the \texttt{\_printbib} sets the indentation by \texttt{\hangindent} and prints numeric \{bib-marks\} by \llap{\{\the\bibnum\}}. If \texttt{\nonumcitations} then the \texttt{\_citelinkA} is not empty and \{bib-marks\} (\texttt{\the\bibnum} nor \texttt{\the\bibmark}) are not printed. The text of bib-entry follows. User can create this text manually using \texttt{\bib} command or it is generated automatically from a .bib database by \texttt{\usebib} command.

The vertical space between bib-entries is controlled by \texttt{\_bibskip} macro.

The \texttt{\usebib} command is implemented in \texttt{usebib.ompl} file which is loaded when the \texttt{\usebib} command is used first. The \texttt{usebib.ompl} file loads the \texttt{librarian.tex} for scanning the .bib files. See the section \texttt{2.32.2}, where the file \texttt{usebib.ompl} is documented.

\nobibwarning\{\{list of bib-labels\}\} declares a list of bib labels which are not fully declared in .bib file but we want to suppress the warning about it. List of bib labels are comma-separated case sensitive list without spaces.
2.32.2 The \usebib command

The file \texttt{usebib.opm} implements the command \texttt{\usebib/(sorttype) ⟨(style)⟩ ⟨bibfiles⟩} where ⟨sorttype⟩ is one letter \texttt{c} (references ordered by citation order in the text) or \texttt{s} (references ordered by key in the style file), ⟨style⟩ is the part of the name \texttt{bib-⟨(style)⟩.opm} of the style file and ⟨bibfiles⟩ are one or more \texttt{.bib} file names without suffix separated by comma without space. Example:

\texttt{\usebib/s (simple) mybase, yourbase}

This command reads the ⟨bibfiles⟩ directly and creates the list of bibliographic references (only those declared by \texttt{\cite} or \texttt{\nocite} in the text). The formatting of such references is defined in the style file.

The principle “first entry wins” is used. Suppose \texttt{\usebib/s (simple) local, global}. If an entry with the same label is declared in \texttt{local.bib} and in \texttt{global.bib} too then the first wins. So, you can set exceptions in your \texttt{local.bib} file for your document.

The \texttt{bib-⟨(style)⟩.opm} declares entry types (like \texttt{@BOOK, @ARTICLE}) and declares their mandatory and optional fields (like \texttt{author, title}). When a mandatory field is missing in an entry in the \texttt{.bib} file then a warning is printed on the terminal about it. You can suppress such warnings by command \texttt{\nobibwarning \{⟨bib-labels⟩\}}, where ⟨bib-labels⟩ is a comma-separated list of labels (without spaces) where missing mandatory fields will be no warned.

Old \texttt{.bib} files may use the obscure notation for accents like \texttt{\textbackslash"o}. Recommendation: convert such old files to Unicode encoding. If you are unable to do this then you can set \texttt{\bibtexhook={\oldaccents}}.

2.32.3 Notes for bib-style writers

The \texttt{.bib} files include records in the format:

\begin{verbatim}
&⟨entry-type⟩\{⟨label⟩, \\
⟨field-name⟩ = "⟨field-data⟩", \\
⟨field-name⟩ = "⟨field-data⟩", \\
... etc
\end{verbatim}

see the file \texttt{demo/op-biblist.bib} for a real example. The ⟨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 \url{http://en.wikipedia.org/wiki/BibTeX}. The set of entry types listed on this \url{http://en.wikipedia.org/wiki/BibTeX} 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 \texttt{.bib} records.

The style writer can be inspired by two styles already done: \texttt{bib-simple.opm} and \texttt{bib-iso690.opm}. The second one is documented in detail in section 2.32.5.

The printing rules for each ⟨entry-type⟩ must be declared by \texttt{\_sdef\_print:⟨entry-type⟩} in \texttt{bib-⟨(style)⟩.opm} file. The ⟨entry-type⟩ has to be lowercase here. Op\TeX{} supports following macros for a more comfortable setting of printing rules:

\begin{itemize}
  \item \texttt{\_bprinta \{⟨field-name⟩\} \{⟨if defined⟩ \{⟨if not defined⟩\}. The part ⟨if defined⟩ is executed if ⟨field-name⟩ is declared in \texttt{.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⟩.
  \item The part ⟨if not defined⟩ can include the \texttt{\bibwarning} command if the ⟨field-name⟩ is mandatory.
  \item \texttt{\_bprintb \{⟨field-name⟩\} \{⟨if defined⟩ \{⟨if not defined⟩\}. The same as \texttt{\_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 \texttt{\_bprintb} commands are nested (\texttt{\_bprintb in \_bprintb}), then you need to write the \#\#\#\#1 parameter for internal \texttt{\_bprintb}. But if \texttt{\_bprinta} commands are nested then the parameter is not duplicated.
  \item \texttt{\_bprintf \macro \{⟨if non-empty⟩\}. The ⟨if non-empty⟩ part is executed if \texttt{\macro} is non-empty. The * parameter can be used, it is replaced by the \texttt{\macro}.
\end{itemize}

164
\_\_printv \{\langle field1 \rangle, \langle field2 \rangle, \ldots \} \{\{if defined\}\} \{\{if not defined\}\}. The part \{if defined\} is executed if \langle field1 \rangle or \langle field2 \rangle or ... is defined, else the second part \{if not defined\} is executed. There is one filed name or the list field names separated by commas. The parts cannot include any parameters.

There are two special field-names: \_author and \_editor. The processed list of authors or editors are printed here instead of raw data, see the commands \_authorname and \_editorname below.

The bib-style writer can define \_print:BEGIN and/or \_print:END. They are executed at the beginning or end of each \langle entry-type \rangle. 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 \verb!\cite! marks when \verb!\nonumcitations! is set.

Moreover, the bib-style writer must declare the format of special fields author and editor. These fields include a list of names, each name is precessed individually in a loop. The \_authorname or \_editorname is called for each name on the list. The bib-style writer must define the \_authorname and \_editorname commands in order to declare the format of printing each individual name. The following control sequences can be used in these macros:

- \_\_NameCount: the number of the currently processed author in the list
- \_\_namecont: the total number of the authors in the list
- \_\_Lastname, \_\_Firstname, \_\_Von, \_\_Junior: the parts of the name.

The whole style file is read in the group during the \_\_usebib command is executed before typesetting the reference list. Each definition or setting is local here.

The auto-generated phrases (dependent on current language) can be used in bib-style files by \_\_mtext{\verb!bib.\{identifier\}!}, where \{ident\} is an identifier of the phrase and the phrase itself is defined by \_\_sdef{\verb!\_mt:bib.\{identifier\}:\{language\}!}{\{phrase\}}. See section 2.37.2 for more detail. Phrases for \{identifiers\}: and, etal, edition, citedate, volume, number, prepages, postpages, editor, editors, available, availablealso, bachthesis, masthesis, phdthesis are defined already, see the end of section 2.37.2.

If you are using non-standard field-names in .bib database and bib-style, you have to declare them by \_\_CreateField \{\langle fieldname \rangle\}. You can declare \_\_SortingOrder in the manner documented by librarian package.

User or author of the bib-style can create the hidden field which has a precedence while sorting names. Example:

\_\_CreateField \{\langle fieldname \rangle\}
\_\_SpecialSort \{\langle fieldname \rangle\}

Suppose that the .bib file includes:

... \author = "Jan Chadima",
\_\_specialsort = "\_\_Lastname, \_\_Firstname",
... \author = "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 \_\_specialsort = "\_\_". because this character is sorted before A.

### 2.32.4 The usebib.\_\_opm macro file loaded when \_\_usebib is used

Loading the librarian.\_\_tex macro package. See texdoc librarian for more information about it.

We want to ignore \texttt{\_\_errmessage} and we want not to create \jobname.\_\_lbr file.

---

\texttt{\_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{3 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{3 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{4 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{5 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{6 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{7 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{8 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{9 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{10 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{11 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{12 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{13 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{14 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{15 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{16 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{17 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{18 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{19 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}

\texttt{20 \_\_codedecl \MakeReference \{Reading bib databases <2022-02-04>\} \% loaded on demand by \_\_usebib}
The \usebib command.

\begin{verbatim}
\def\usebib/#1 (#2) #3 {% 
\let\citelI\relax \xdef\citelist{} \global \ea\let\csname _ctlst:\_bibp\endcsname =\write \ifx\citelist\empty \opwarning{No cited items. \noexpand\usebib ignored}% 
\else 
\bgroup \par \endemergencystretch=.3\hsize \def\optexbibstyle{#2} \setctable\optexcatcodes \input bib-#2.opm \the\bibtexhook \ifcsname _mt:bib.and:\_cs{_lan:\the\language}\endcsname \else \opwarning{\usebib: No phrases for language \"\cs{_lan:\the\language}\" (using "en")}% \fi \def\tmp##1[*]\relax{\def\tmp{##1}}\ea\tmp\citelist[*]\relax \ifx\tmp\empty\else % there was \nocite[*] used. \setbox0=\vbox{\hsize=\maxdimen \def\citelist{}\adef@{\readbibentry}\input #3.bib \ea}\ea\def\ea\citelist\ea{\citelist} \fi \def\citeI[#1]{\csname lb@cite\endcsname{##1}{\bibp}{}{}} \citelist \BibFile{#3} \if s#1\SortList{\bibp}\fi \ReadList{} \restorectable \egroup \fi 
\def\readbibentry#1#{\readbibentryA} \def\readbibentryA#1{\readbibentryB#1,,\relax!.} \def\readbibentryB#1#2,#3\relax!.{\addto\citelist{\citeI[#1#2]}}
\end{verbatim}

Corrections in librarian macros.

\begin{verbatim}
\tmpnum=\catcode`\@=11 \def\lb@checkmissingentries#1,{% we needn't \errmessage here, only \opmacwarning \unless\ifx\lb@temp\lb@eoe \lb@next \lb@ifcs\lb@temp{requested}{\let\lb@entrykey\lb@temp \lb@ifcs\lb@entrykey{fields}{}{\lb@defcs\lb@entrykey{fields}{}\lowercase{\lb@addfield{entrytype}{#1}}\let\lb@next\lb@analyzeentry}}{} \lb@next \fi }% \\def\lb@readentry#1#2#3,{% space before key have to be ignored \def\lb@btemp[#2]\lb@eoe % we need case sensitive keys \unless\ifx\lb@btemp\lb@eoe \lb@ifcs\lb@btemp{fields}{} \lb@defcs\lb@btemp{fields}{} \\lowercase{\lb@addfield{entrytype}{#1}}\let\lb@next\lb@analyzeentry}{} \\let\lb@compareA=\lb@compare \\let\lb@preparesortA=\lb@preparesort \def\lb@compare#1\lb@eoe#2\lb@eoe{% SpecialSort: \ifx\lb@sorttype\lb@namestring \\ifx\sortfield\undefined\lb@compareA#1\lb@eoe#2\lb@eoe \else \lb@defcs\lb@entrykey{fields}{} \\lowercase{\lb@addfield{entrytype}{#1}}\let\lb@next\lb@analyzeentry}{} \\let\lb@next \let\lb@bcompareA=\lb@bcompare \let\lb@bpreparesortA=\lb@bpreparesort \def\lb@bcompare#1\lb@eoe#2\lb@eoe{% SpecialSort: \ifx\lb@sorttype\lb@namestring \\lb@bRetrieveFieldInFor\lb@eoe{\sortfield}\lb@bentrykey{\lb@btemp}\lb@eoe \\ifx\lb@btemp\empty\{\let\lb@next\lb@banalyzeentry}{} \\lb@next \\let\lb@bcompareA=\lb@bcompare }
\end{verbatim}

166
Main action per each entry.

\newcommand{\MakeReference}{\par \bibskip
  \if\isdefined{\_bim:\_bibp\the\bibnum}\ift\else
    \edef\tmpb{\_csname _bim:\_bibp\the\bibnum\_endcsname}\%
    \bibmark=\_ea{\_tmpb}\%
    \else \bibmark={}\fi
  
  \edef\tmpb{\EntryKey}\%
  \noindent \_dbib\EntryKey
  \printbib
  {\_RetrieveFieldIn{entrytype}\_entrytype
    \csname _print:BEGIN\_endcsname
      \isdefined{\_print:\_entrytype}\ift\else
        \csname _print:BEGIN\_endcsname
        \else
          \ifx\_entrytype\empty\else
            \_opwarning{Entrytype @\_entrytype\space from \EntryKey\ undefined}\%
          \else
            \_RetrieveFieldIn{\_entrytype}\_bibfield
          \fi
        \fi
      \fi
    \_csname _print:\_entrytype\_endcsname
  \fi
  \_wbib \EntryKey {\the\bibnum}{\the\bibmark}\%
  \par
Various macros + multilingual. Note that \nobibwarnlist is used in \bibwarning and it is set by \nobibwarning macro.

2.32.5 Usage of the bib-iso690 style

This is the iso690 bibliographic style used by OpTeX. See op-biblist.bib for an example of the .bib input. You can try it by:

\usebib[s (iso690) op-biblist]

Common rules in .bib files
There are entries of type @FOO{...} in the .bib file. Each entry consists of fields in the form name=m, "value". or name=m, {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 “ and ”. Each author can be written in various formats (the 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 “ and ” between authors will be converted to comma during printing, but between the semifinal and final author the word “and” (or something different depending on the current language) is printed.

The first author is printed in reverse order: “LASTNAME, Firstname(s) von, After” and the other authors are printed in normal order: “Firstname(s) von LASTNAME, After”. This feature follows the ISO 690 norm. The Lastname is capitalized using uppercase letters. But if the \caps font modifier is defined, then it is used and printed {\caps\_rm Lastname}.

You can specify the option aumax:⟨number⟩. The ⟨number⟩ denotes the maximum authors to be printed. The rest of the authors are ignored and the et-al. is appended to the list of printed authors. This text is printed only if the aumax value is less than the real number of authors. If you have the same number of authors in the .bib file as you need to print but you want to append et-al. then you can use auetal option.

There is an aumin:⟨number⟩ option which denotes the definitive number of printed authors if the author list is not fully printed due to aumax. If aumin is unused then aumax authors are printed in this case.

All authors are printed if aumax:⟨number⟩ option isn’t given. There is no internal limit. But you can set the global options in your document by setting the \biboptions tokens list. For example:

\biboptions={aumax:7 aumin:1}
% if there are 8 or more authors then only the first author is printed.

Examples:

author = "John Green and Bob Brown and Alice Black",
output: GREEN, John, Bob BROWN, and Alice BLACK.

author = "John Green and Bob Brown and Alice Black",
option = "aumax:1",
output: GREEN, John et al.

author = "John Green and Bob Brown and Alice Black",
option = "aumax:2",
output: GREEN, John, Bob BROWN et al.

author = "John Green and Bob Brown and Alice Black",
option = "aumax:3",
output: GREEN, John, Bob BROWN, and Alice BLACK.

author = "John Green and Bob Brown and Alice Black",
option = "auetal",
output: GREEN, John, Bob BROWN, Alice BLACK et al.

If you need to add a text before or after the author’s list, you can use the auprint:⟨value⟩ option. The ⟨value⟩ will be printed instead of the authors list. The ⟨value⟩ can include \AU macro which expands to the authors list. Example:

author = "Robert Calbraith",
option = "auprint:\{AU\space [pseudonym of J. K. Rowling]\}"
output: CALBRAITH Robert [pseudonym of J. K. Rowling].

You can use the autrim:⟨number⟩ option. All Firstnames of all authors are trimmed (i. e. reduced to initials) iff the number of authors in the author field is greater than or equal to ⟨number⟩. There is
an exception: `autrim:0` means that no Firstnames are trimmed. This is the default behavior. Another example: `autrim:1` means that all Firstnames are trimmed.

```latex
\begin{verbatim}
author = "John Green and Bob Brown and Alice Black",
option = "auetal autrim:1",
\end{verbatim}
```

output: GREEN, J., B. BROWN, A. BLACK et al.

If you need to write a team name or institution instead of authors, replace all spaces by \`\textbackslash\` in this name. Such text is interpreted as Lastname. You can add the secondary name (interpreted as Firstname) after the comma. Example:

```latex
\begin{verbatim}
author = "Czech\ Technical\ University\ in\ Prague, Faculty\ of\ Electrical\ Engineering",
\end{verbatim}
```

output: CZECH TECHNICAL UNIVERSITY IN PRAGUE, Faculty of Electrical Engineering.

**The editor field**
The editor field is used for the list of the authors of the collection. The analogous rules as in author field are used here. It means that the authors are separated by “and”, the Firstnames, Lastnames, etc. are interpreted and you can use the options `edmax:⟨number⟩`, `edmin:⟨number⟩`, `edetal`, `edtrim:⟨number⟩` and `edprint:{⟨value⟩}` (with \ED macro). Example:

```latex
\begin{verbatim}
editor = "Jan Tomek and Petr Karas",
option = "edprint:\{\ED, editors.\} edtrim:1",
\end{verbatim}
```

Output: J. TOMEK and P. KARAS, editors.

If `edprint` option is not set then `{\ED, \text{eds.}}` or `{\ED, \text{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 edition info. The value is read as raw data without any interpretation of Lastname, Firstname etc.

```latex
\begin{verbatim}
ednote = "Illustrations by Robert \upper{Agarwal}, edited by Tom \upper{Nowak}"
\end{verbatim}
```

output: Illustrations by Robert AGARWAL, edited by Tom NOWAK.

The `\upper` command has to be used for Lastnames in the ednote field.

**The title field**
This is the title of the work. It will be printed (in common entry types) by italics. The ISO 690 norm declares, that the title plus optional subtitle are in italics and they are separated by a colon. Next, the optional secondary title has to be printed in an upright font. This can be added by `titlepost:{⟨value⟩}`. Example:

```latex
\begin{verbatim}
title = "The Simple Title of The Work",
or
\end{verbatim}
```

or

```latex
\begin{verbatim}
title = "Main Title: Subtitle",
\end{verbatim}
```

or

```latex
\begin{verbatim}
title = "Main Title: Subtitle",
option = "titlepost:{Secondary title}";
\end{verbatim}
```

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:

```latex
\begin{verbatim}
edition = "Second",
edition = "2nd",
edition = "2\$\{\text{rd}\}\$",
edition = "2.\",
\end{verbatim}
```

Output of the last example: 2. ed.
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:{{\em\value}} option. The the \value is printed instead of edition field and shortcut. The edition field must be set. Example:

\begin{verbatim}
edition = "whatever",
option = "editionprint:{Second full revised edition}";
\end{verbatim}


You can use \texttt{\textbackslash EDN} macro in \texttt{editionprint} value. This macro is expanded to the edition value. Example:

\begin{verbatim}
edition = "Second",
option = "editionprint:{\texttt{\textbackslash EDN} space full revised edition}";
or
edition = "Second full revised edition",
option = "editionprint:{\texttt{\textbackslash EDN}}",
\end{verbatim}

\textbf{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 noautoletters option.

You can use \"yearprint:\{\em\value\}" option. If it is set then the \em\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}


year = "2012a",
option = "yearprint:{2012}"


The address, publisher, and year are typically mandatory fields. If they are missing then the warning occurs. But you can set unpublished option. Then this warning is suppressed. There is no difference in the printed output.

\textbf{The url field}

Use it without \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:


If the cs language is the current one than the output is:

Dostupné z: http://petr.olsak.net/opmac.html.

If the urlalso option is used, then the added text has the form “Available also from” or “Dostupné také z:” (if cs language is current).

\textbf{The citedate field}

This is the citation date. The field must be in the form year/month/day. It means, that the two slashes must be written here. The output depends on the current language. Example:
citedate = "2004/05/21",
Output when en is current: [cit. 2004-05-21].
Output when cs is current: [vid. 21. 5. 2004].

The howpublished field
This declares the available medium for the cited document if it is not in printed form. Alternatives: online, CD, DVD, etc. Example:

howpublished = "online",
Output: [online].

The volume, number, pages and numbering fields
The volume is the “big mark” of the journal issue and the number is the “small mark” of the journal issue and pages includes the page range of the cited article in the journal. The volume is prefixed by Vol., the number by No., and the pages by pp. But these prefixes depends on the language of the entry.
Example:

volume = 31,
number = 3,
pages = "37--42",

volume = 31,
number = 3,
pages = "37--42",
lang = "cs",
Output: ročník 31, č. 3, s. 37–42.

If you disagree with the default prefixes, you can use the numbering field. When it is set then it is used instead of volume, number, pages and instead of any mentioned prefixes. The numbering can include macros VOL, NO, PP, which are expanded to the respective values of fields. Example:

volume = 31,
number = 3,
pages = "37--42",
numbering = "Issue~\VOL/\NO, pages~\PP",

Note: The volume, numbers, and pages fields are printed without numbering filed only in the @ARTICLE entry. It means, that if you need to visible them in the @INBOOK, @INPROCEEDINGS etc. entries, then you must use the numbering field.

Common notes about entries
The order of the fields in the entry is irrelevant. We use the printed order in this manual. The exclamation mark (!) denotes the mandatory field. If the field is missing then a warning occurs during processing.

If the unpublished option is set then the fields address, publisher, year, isbn, and pages are not mandatory. If the nowarn option is set then no warnings about missing mandatory fields occur.

If the field is used but not mentioned in the entry documentation below then it is silently ignored.

- The @BOOK entry
  This is used for book-like entries.
  Fields: author(!), title(!), howpublished, edition, ednote, address(!), publisher(!), year(!), citedate, series, isbn(!), doi, url, note.
  The ednote field here means the secondary authors (illustrator, cover design etc.).

- The @ARTICLE entry
  This is used for articles published in a journal.
  Fields: author(!), title(!), journal(!), howpublished, address, publisher, month, year, [numbering or volume, number, pages(!)], citedate, issn, doi, url, note.
  If the numbering is used then it is used instead volume, number, pages.

- The @INBOOK entry
This is used for the part of a book. The author field is used for the student’s thesis.
Fields: author(!), title(!), howpublished, address(!), month, year(!), citedate, type(!), ednote, doi, url, note.

There are nearly equivalent entries: @BACHELORSTHESIS, @MASTERSTHESIS and @PHDTHESIS. These entries set the type field to an appropriate value automatically. The type field is optional in this case. If it is used then it has precedence before the default setting.

• The @MISC entry
It is intended for various usage.
Fields: author, title, howpublished, ednote, citedate, doi, url, note.

You can use \AU, \ED, \EDN, \VOL, \NO, \PP, \ADDR, \PUBL, \YEAR macros in ednote field. These macros print authors list, editors list, edition, volume, number, pages, address, publisher, and year field values respectively.

The reason for this entry is to give to you the possibility to set the format of entry by your own decision. The most of data are concentrated in the ednote field.

• The @BOOKLET, @INCOLLECTION, @MANUAL, @PROCEEDINGS, @TECHREPORT, @UNPUBLISHED entries
These entries are equivalent to @MISC entry because we need to save the simplicity. They are implemented only for (almost) backward compatibility with the ancient BibTEX. But the ednote is mandatory field here, so you cannot use these entries from the old databases without warnings and without some additional work with the .bib file.

The cite-marks (bibmark) used when \nonumcitations is set
When \nonumcitations is set then \cite prints text-oriented bib-marks instead of numbers. This style file auto-generates these marks in the form “Lastname of the first author, comma, space, the year” if the bibmark field isn’t declared. If you need to set an exception from this common format, then you can use bibmark field.

The OPmac trick http://petr.olsak.net/opmac-tricks-e.html#bibmark describes how to redefine the algorithm for bibmark auto-generating when you need the short form of the type [Au13].

Sorting
If \usebib/c is used then entries are sorted by citation order in the text. If \usebib/s is used then entries are sorted by “Lastname, Firstname(s)” of the first author and if more entries have this value equal, then the year is used (from older to newer). This feature follows the recommendation of the ISO 690 norm.

If you have the same authors and the same year, you can control the sorting by setting years like 2013, 2013a, 2013b, etc. You can print something different to the list using yearprint{⟨value⟩} option, see the section about address, publisher, and year above. The real value of year field (i.e. not yearprint value) is also used in the text-oriented bib-marks when \nonumcitations is set.

If you have some problems with name sorting, you can use the hidden field key, which is used for sorting instead of the “Lastname Firstname(s)” of authors. If the key field is unset then the “Lastname Firstname(s)” is used for sorting normally. Example:

    author = "Světla Čmejrková",
    key = "Czzmejrkova Svetla",

This entry is now sorted between C and D.

The norm recommends placing the auto-citations at the top of the list of references. You can do this by setting key,=,"@", to each entry with your name because the @ character is sorted before A.
Languages
There is the language of the outer document and the languages of each entry. The ISO 690 norm recommends that the technical notes (the prefix before URL, the media type, the “and” conjunction between the semifinal and final author) maybe printed in the language of the outer document. The data of the entry have to be printed in the entry language (edition ed./vyd., Vol./ročník, No./č. etc.). Finally, there are the phrases independent of the language (for example In:). Unfortunately, the bib\TeX supposes that the entry data are not fully included in the fields so the automaton has to add some text during processing (“ed.”, “Vol.”, “see also”, etc.). But what language has to be chosen?

The current value of the \texttt{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 \texttt{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 \texttt{lang} is not set then the outer document language is used.

You can use \texttt{\_Mtext(bib.(identifier))} if you want to use a phrase dependent on outer document language (no on entry language). Example:

\begin{verbatim}
howpublished = "\_Mtext(bib.blue-ray)"
\end{verbatim}

Now, you can set the variants of \texttt{bib.blue-ray} phrase for various languages:

\begin{verbatim}
\_sdef\{mt:bib.blue-ray:en\} {Blue-ray disc}
\_sdef\{mt:bib.blue-ray:cs\} {Blue-ray disk}
\end{verbatim}

Summary of non-standard fields
This style uses the following fields unknown by bib\TeX:

\begin{verbatim}
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
\end{verbatim}

Summary of options

\begin{verbatim}
aumax:{\langle number\rangle} ... maximum number of printed authors
aumin:{\langle number\rangle} ... number of printed authors if aumax exceeds
autrim:{\langle number\rangle} ... full Firstnames iff number of authors are less than this
auprint:{\langle value\rangle} ... text instead authors list (\texttt{\AU} macro may be used)
edmax, edmin, edtrim ... similar as above for editors list
edprint:{\langle value\rangle} ... text instead editors list (\texttt{\ED} macro may be used)
titlepost:{\langle value\rangle} ... text after title
yearprint:{\langle value\rangle} ... text instead real year (\texttt{\YEAR} macro may be used)
editionprint:{\langle value\rangle} ... text instead of real edition (\texttt{\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
\end{verbatim}

Other options in the option field are silently ignored.

2.32.6 Implementation of the bib--iso690 style
\_maybetod (alias \: in the style file group) does not put the second dot.

Option field.

Formating of Author/Editor lists.
Preparing bib-mark (used when \nonumcitations). The \setbibmark is run at the end of each record. The \authlist includes Lastname,Firstname,Von,Junior of all authors separated by semicolon (no semicolon at the end of the list). If bibmark isn’t declared explicitly then we create it by the \createbibmark⟨year⟩;⟨authors-list⟩;;;;\fin macro. It outputs first Lastname (and adds “et al.” if the second author in the ⟨authors-list⟩ is non-empty). Then comma and ⟨year⟩ is appended. A user can redefine the \createbibmark macro in the \bibtexhook tokens list, if another bibmark format is needed. The macro \createbibmark must be expandable. See also OpTeX trick 0104.

\setbibmark{\ifx\authlist\undefined \def\authlist{,;}\fi \RetrieveFieldIn{bibmark}\tmp \ifx\tmp\empty \RetrieveFieldIn{year}\tmp \edef\tmp{\createbibmark\expanded{\tmp;\authlist};,;,;,;\fin}\fi \bibmark=\ea\tmp%}{\ifnum0\namecount<\autrim\relax \AbbreviateFirstname \fi}

Setting phrases.

\chardef\documentlanguage=\language
\def\Mtext#1{\ifnum0\namecount<1 \def\cnamex_{#1}\fi}
\def\CreateField {ednote}
\def\CreateField {citedate}
\def\CreateField {numbering}
\def\CreateField {isbn}
\def\CreateField {issn}
\def\CreateField {doi}
\def\CreateField {url}
\def\CreateField {bibmark}

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. The sorting can be used for various other applications, see an example in Op\TeX\ trick 0068.

There are two passes in the sorting algorithm. The primary pass does not distinguish between a group of letters (typically non-accented and accented). If the result of comparing two string is equal in primary pass then the secondary pass is started. It distinguishes between variously accented letters. Czech rules, for example, says: not accented before dieresis before acute before circumflex before ring. At less priority: lowercase letters must be before uppercase letters.

The \_\sortingdatalatin implements these rules for the languages with latin alphabets. The groups between commas are not distinguished in the first pass. The second pass distinguishes all characters mentioned in the \_\sortingdatalatin (commas are ignored). The order of letters in the \_\sortingdatalatin macro is significant for the sorting algorithm.
Characters to be ignored during sorting are declared in \_ignoredcharsgeneric. These characters are ignored in the first pass without additional condition. All characters are taken into account in the second pass: ASCII characters with code < 65 are sorted first if they are not mentioned in the \_sortingdata\... macro. Others not mentioned characters have undefined behavior during sorting. Sorting is always processed by rules of a given language. The macros \_sortingdata\langle lang-tag\rangle, \_ignoredchars\langle lang-tag\rangle and \_compoundchars\langle lang-tag\rangle declare these rules. The \langle lang-tag\rangle is ISO code of the language: en, cs, de, pl, es for example. The English language is implemented here. Other languages are implemented in the lang-data.opm file (see section 2.37.4). The \_compoundchars\langle lang-tag\rangle can declare changes performed before sorting. For example Czech language declares:

\_let \_sortingdatacs = \_sortingdatalatin % Czech alphabet is subset of Latin
\_def \_compoundcharscs {ch:^^T Ch:^^U CH:^^V}

It transforms two-letters ch to single character ^ ^ T because ch is treated as single compound character by Czech rules and CH is sorted between H and I. See \_sortingdatalatin where ^ ^ T is used. This declaration makes more transformations of Ch and CH too. The declarations of the form x:y in the \_compoundchars\langle lang-tag\rangle are separated by space.

You can declare a transformation from single letter to more letters too. For example German rules sets ß equal to ss during sorting:

\_let \_sortingdatade = \_sortingdatalatin % German alphabet is subset of Latin
\_def \_compoundcharsde {ß:ss}

If there are two words equal after first pass of sorting: Masse (mass) and Maße (measures) for example, then second pass must decide about the order. DIN 5007, section 6.1 says: ss must be before ß in this case. So, we want to switch off the \_compoundchars declaration for the second pass and use the order of s and ß given in \_sortingdata. This is possible if the \_xcompoundchars\langle lang-tag\rangle is defined. It has precedence in the second pass of sorting. We declare for German:

\_def \_xcompoundcharsde {}

German rules mention alternative sorting for phone-books or similar lists of names. The letters ä ö ü should be interpreted as ae, oe and ue. So we get Mueller < Müller < Muff. If this rule is not taken into account, we get Mueller < Muff < Müller. The rule can be implemented by:


Because ù < ū in \_sortingdata and because \_xcompoundcharsde is empty, we have Mueller < Müller after second pass of the sorting.

You can declare these macros for more languages if you wish to use \_makeindex with sorting rules with respect to your language. Note: if you need to map compound characters to a character, don’t use ""I, ""J or ""M because these characters have very specific category codes.

If you created \_sortingdata etc. for your language, please, send them to me. I am ready to add them to the file lang-data.opm in a new OpTeX release. See also section 2.37.4.
French sorting rule says: if the words are the same except for accents then accented letters are sorted after unaccented letters but read the words from their end in the second pass. For example correct sorting is: cote < côte < coté < côté. This rule can be activated if the control sequence \secondpass\reversewords is set to \secondpassfr. For example, lang-data.om declares \secondpassfr=\reversewords.

Preparing to primary pass is performed by the \setprimarysorting macro implemented here. The \tag{lang} is saved to the \primarylang macro when sorting is initialized in \dosorting (it is typically derived from current \language value). The \setprimarysorting is called from \dosorting macro and all processing of sorting is in a group. It sets actual \primarydata, \primarycompoundchars and \primaryignoredchars if given language declares them. If not then warning will be printed using \nold macro and English data are used. The \lccode of all characters from \primarydata and \primaryignoredchars are set. The sorted words will be converted using \primarycompoundchars followed by \lowercase before first pass is run.

\begin{lstlisting}[language=TeX]
\def\setprimarysorting {%
 \ealet \ea\primarydata \csname _primarydata_{\primarylang}\endcsname
 \ealet \ea\primarycompoundchars \csname _primarycompoundchars_{\primarylang}\endcsname
 \ealet \ea\primaryignoredchars \csname _primaryignoredchars_{\primarylang}\endcsname
 \def\nold{}%
 \ifx \primarydata\relax \addto\nold{ primarydata}%
 \let \primarydata = \primarydataen \fi
 \ifx \primarycompoundchars\relax \addto\nold{ compoundchars}%
 \let \primarycompoundchars = \primarycompoundcharen \fi
 \ifx \primaryignoredchars\relax \addto\nold{ ignoredchars}%
 \let \primaryignoredchars = \primaryignoredcharen \fi
 \ifx\nold\empty\else \opwarning{Missing nold space for language (\primarylang)}\fi
 \ifx \primarycompoundchars\empty\else
 \edef \primarycompoundchars {\detokenize\ea{\primarycompoundchars}} \fi % all must be catcode 12
 \def \act ##1{\ifx##1\relax \else
 \ifx##1,\else \advance\tmpnum by1 \lccode`##1=\tmpnum \fi
 \fi}
 \tmpnum=65 \ea\act \primarydata \relax
 \def \act ##1{\ifx##1\relax \else
 \lccode`##1=`\^^I
 \ea\act \fi}
 \ea\act \primaryignoredchars \relax
}
\end{lstlisting}

Preparing to secondary pass is implemented by the \setsecondarysorting macro.

\begin{lstlisting}[language=TeX]
\def\setsecondarysorting {%
 \edef \act ##1{\ifx##1\relax \else
 \ifx##1,\else \advance\tmpnum by1 \lccode`##1=\tmpnum \fi
 \fi}
 \tmpnum=64 \ea\act \primarydata \relax
}
\end{lstlisting}

Strings to be sorted are prepared in \string control sequences (to save \TeX memory). The \preparestring \string converts \string to \tmpb with respect to the data initialized in \setprimarysorting or \setsecondarysorting. The compound characters are converted by the \dcompound macro.

The compoud characters are converted by the \dcompound macro.

\begin{lstlisting}[language=TeX]
\def \preparestring \string #1{%
 \edef \tmpb {\ea\ignoreit\csstring #1}% \,<string> -> <string>
 \ea \dcompound \primarycompoundchars \relax(); % replace compound characters
 \lowercase \ea\def \ea\tmpb \ea\tmpb(\tmppb)\% convert in respect to \primarydata
 \ea\replstring \ea\tmpb \ea(\csstring{"}{})\% remove ignored characters
 \}
\end{lstlisting}

Macro \isAleB \string returns the result of comparison of given two strings to \ifAleB control sequence. Usage: \isAleB \string, \string convert strings (in respect of the data prepared for first pass) must be saved as values of \string and \string macros. The reason is speed: we don’t want to convert them repeatedly in each comparison.

\begin{lstlisting}[language=TeX]
\def \isAleB \string #1\string #2 %{\ifAleB \string #1\string #2}{\fi}
\end{lstlisting}
The macro \_testAleB does the real work. It reads the first character from both converted strings, compares them and if it is equal then calls itself recursively else gives the result.

The \_testAleBSecondary 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 \_mergesort macro expects that all items in \_iilist are separated by a comma when it starts. It ends with sorted items in \_iilist without commas. So \_dosorting macro must prepare commas between items.
The \_dosorting \_list macro redefines \_list as sorted \_list. The \_list have to include control sequences in the form \langle \text{c} \rangle \langle \text{string} \rangle. These control sequences will be sorted with respect to \langle \text{strings} \rangle without change of meanings of these control sequences. Their meanings are irrelevant when sorting. The first character \langle \text{c} \rangle in \langle \text{c} \rangle \langle \text{string} \rangle should be whatever. It does not influence the sorting. OpTeX uses comma at this place for sorting indexes: \langle \text{word1} \rangle \langle \text{word2} \rangle \langle \text{word3} \rangle ... .

The current language (chosen for hyphenation patterns) is used for sorting data. If the macro \_sortinglang is defined as \langle \text{lang-tag} \rangle (for example \_def \_sortinglang{de} for German) then this has precedence and current language is not used. Moreover, if you specify \_asciisortingtrue then ASCII sorting will be processed and all language sorting data will be ignored.

French rules needs reverse reading the words in the second pass. The \_reversewords is activated in this case and it adds new job to the macro \_prepsort: it reverses the letters in the compared words (saved in \_tmpa and \_tmpb) by the expandable \_sortrevers macro. The \_prepsort macro is used in the \_testAleBsecondary and it is empty by default.

The \_makeindex prints the index. First, it sorts the \_iilist second, it prints the sorted \_iilist, each item is printed using \_printindexitem. We set \_leftskip=\iindent and we suppose that each index entry starts by \_noindent\hskip=\iindent (see the macro \_printii). Then the next lines of the same index entry (if the page list is broken to more pages) is indented by \_leftskip=\iindent.
The \_printindexitem \langle word \rangle prints one item to the index. If \langle word \rangle is defined then this is used instead real \langle word \rangle (this exception is declared by iiis 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 ⟩/⟨ sub-word ⟩/⟨ sub-sub-word ⟩. The \everyii tokens register is applied before \noindent. User can declare something special here.

\_definefirstii \langle word \rangle macro defines \_firstii which is used as the \langle letter ⟩ parameter of the macro \_newiiletter and for testing if the “first letter” of the index entry was changed. The \uppercase of the real first letter is used by default here. You can re-implement \_definefirstii if you want. For example, you want to ignore accents above letters for index sub-headers:

\def \_definefirstii #1#2&{
  \uppercase{\def \_firstii{#1}}
}
\def \_iiemdash{\kern.1em---\space}
\def \_lastii{}
\def \_newiiletter#1#2{}
\def \_scanprevii#1/#2&{
  \def \_previi{#2}
  \edef \_tmpa{\detokenize{#1}}
  \ifx \_tmpa \_previi \iiemdash \else \_gdef \_previi{} \fi
}
\def \_printiiA #1/{
  \if^#1^\let \_previi=\_currii \else
    \edef \_tmpb{\detokenize{#1}}
    \ifx \_tmpa \_tmpb \iiemdash \else \_edef \_previi{} \fi
  \fi
}
\def \_printiipages #1&{
  \let \_pgtype=\undefined \_tmpnum=0 \_printpages #1,:,\par
  \_definefirstii \_firstii\_lastii\_previi\_pgtype\_pgpageno
  \edef \_tmpa{\_detokenize{\_firstii}}
  \_gdef \_currii{\_firstii}\_the \_everyii\_noindent
  \hskip\_iindent \ignorespaces \_printiiA #1}

\_printiipages \langle pglist \rangle & gets \langle pglist \rangle in the form \langle pg ⟩:⟨type ⟩, \langle pg ⟩:⟨type ⟩, ... \langle pg ⟩:⟨type ⟩ and it converts them to \langle pg ⟩, \langle pg ⟩, \langle from ⟩--\langle to ⟩, \langle pg ⟩ etc. The same pages must be printed only once and continuous consequences of pages must be compressed to the form \langle from ⟩--\langle to ⟩. Moreover, the consequence is continuous only if all pages have the same \langle type ⟩. Empty \langle type ⟩ is most common, pages with b \langle type ⟩ must be printed as bold and with i \langle type ⟩ as italics. Moreover, the \langle pg ⟩ mentioned here are \langle pageno ⟩, but we have to print \langle pageno ⟩. The following macros solve these tasks.
You can re-define \_pgprint ⟨gpageno⟩:{⟨iitype⟩} if you need to implement more ⟨iitypes⟩.

The \_index{⟨word⟩} puts one ⟨word⟩ to the index. It writes \_Xindex{⟨word⟩}{⟨iitype⟩} to the .ref file. All other variants of indexing macros expand internally to \index.

The \_Xindex{⟨word⟩}{⟨iitype⟩} stores ⟨word⟩ to the \_iilist if there is the first occurrence of the ⟨word⟩. The list of pages where ⟨word⟩ occurs, is the value of the macro \_⟨word⟩, so the ⟨gpageno⟩:{⟨iitype⟩} is appended to this list. Moreover, we need a mapping from ⟨gpageno⟩ to ⟨pageno⟩, because we print ⟨pageno⟩ in the index, but hyperlinks are implemented by ⟨gpageno⟩. So, the macro \_pgi:{⟨gpageno⟩} is defined as ⟨pageno⟩.

The implementation of macros \_ii, \_iid, \_iis follows. Note that \_ii works in the horizontal mode in order to the \write whatsis is not broken from the following word. If you need to keep vertical mode, use \iiindex{⟨word⟩} directly.

The \_iitype {⟨type⟩} saves the ⟨type⟩ to the \_iitypesaved macro. It is used in the \_iindex macro.
2.34 Footnotes and marginal notes

\fnotenum is a counter which counts footnotes globally in the whole document.
\lfnotenum is a counter which counts footnotes at each chapter from one. It is used for local page footnote counters too.
\ifpgfnotenum says that footnote numbers are counted on each page from one. We need to run \openref in this case.
\fnotenum is a macro that expands to footnote number counted in declared part.
\fnotenumchapters declares footnotes numbered in each chapter from one (default), \fnotenumglobal declares footnotes numbered in whole document from one and \fnotenumpages declares footnotes numbered at each page from one.

The \printfnotemark prints the footnote mark. You can re-define this macro if you want another design of footnotes. For example

\fnotenumpages
\def \printfnotemark {
\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<0#2 trick returns true if ⟨pageno⟩ has a numeric type and false if it is a non-numeric type (Roman numeral, for example). We prefer to use ⟨pageno⟩, but only if it has the numeric type. We use ⟨gpageno⟩ in other cases.

User can declare \fixmnotes\left or \fixmnotes\right. It defines \_mnotesfixed as \_left or \_right which declares the placement of all marginal notes and such declaration has a precedence.

The \_mnoteD\{⟨text⟩\} macro sets the position of the marginal note. The outer box of marginal note has zero width and zero depth and it is appended after current line using \vadjust primitive or it is inverted to vertical mode as a box shifted down by \parskip and with \vskip\-\baselineskip\vskip\-\parskip followed.

The \_mnoteskip is a dimen value that denotes the vertical shift of marginal note from its normal position. A positive value means shift up, negative down. The \_mnoteskip register is set to zero after the marginal note is printed. The new syntax \mnote up\{dimen\}\{⟨text⟩\} is possible too, but public \_mnoteskip is kept for backward compatibility.
The \_mnoteA macro does the real work. The \_lrmnote\{⟨left⟩\}⟨⟨right⟩\} uses only first or only second parameter depending on the left or right marginal note.

We don’t want to process \note, \fnotemark, \mnote in TOC, headlines nor outlines.

2.35 Styles

OpTeX provides three styles: \report, \letter and \slides. Their behavior is documented in user part of the manual in section1.7.2 and \slides style (for presentations) is documented in op-slides.pdf which is an example of the presentation.

2.35.1 \report and \letter styles

We define auxiliary macro first (used by the \address macro)

The \_boxlines\{⟨line-1⟩\}⟨⟨eol⟩\}⟨⟨line-2⟩\}⟨⟨eol⟩\}...⟨⟨line-n⟩\}⟨⟨eol⟩\} 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 ⟨⟨eol⟩\} is set active and \everypar starts \hbox and active ⟨⟨eol⟩\} closes this \hbox by \).
The \letter style initialization macro is defined here.

The \letter defines \address and \subject macros.

See the files demo/op-letter-*.tex for usage examples.

\def\letter{
  \def\address{\vtop\bgroup\boxlines \parskip=\opt \let\par=\egroup}
  \def\subject{{\bf \mtext{subj}: }}
  \public \address \subject ;
  \typosize[11/14]
  \vsize=\dimexpr \topskip + 49\baselineskip \relax % added 2020-03-28
  \parindent=0pt
  \parskip=\medskipamount
  \nopagenumbers
}
\public \letter \report ;

The \slides macro reads macro file slides.opm, see the section 2.35.2.

\def\slides{\par
  \opinput{slides.opm}
  \adef*(\relax\ifmmode*\else\ea\startitem\fi}
}
\public \slides ;

2.35.2 \slides style for presentations

Default margins and design is declared here. The \ttfont is scaled by mag1.15 in order to balance the ex height of Helvetica (Heros) and LM fonts Typewriter. The \begtt...\endtt verbatim is printed by smaller text.

\margins/1 a5l (14,14,10,3)mm % landscape A5 format
\def\wideformat{\margins/1 (263,148) (16,16,10,3)mm } % 16:9 format
\ifx\fontnamegen\undefined \fontfam[Heros]
\let\ttfont=\undefined \famvardef\ttfont{\setfontsize{mag1.15}\tt}
\fi
\typosize[16/19]
\def\urlfont{ }
\everytt{\typosize[13/16] \advance\hsize by10mm}
\fontdef\fixbf{\bf}
\nopagenumbers
\parindent=0pt
\ttindent=5mm
\parskip=5pt plus 4pt minus2pt
\rightskip=0pt plus 1fil
\ttindent=10pt
\def\ttskip{\smallskip}
\let\scolor=\Blue % secondary color used in default design
\onlyrgb % RGB color space is better for presentations

The bottom margin is set to 3 mm. If we use 1 mm, then the baseline of \footline is 2 mm from the bottom page. This is the depth of the \Grey rectangle used for page numbers. It is r-lapped to \hoffset width because left margin = \hoffset = right margin. It is 14 mm for narrow pages or 16 mm for wide pages.
The \subtit is defined analogically like \tit.

The \pshow{num} prints the text in invisible (transparent) font when \layernum<⟨num⟩. For transparency we need to define special graphics states.

The main level list of items is activated here. The \_item:X and \_item:x are used and are re-defined here. If we are in a nested level of items and \pg^+ is used then \egroups macro expands to the right number of \egroups to close the page correctly. The level of nested item lists is saved to the \_ilevel register and used when we start again the next text after \pg^+.

The default values of \pg, i.e. \pg^; \pg^+ and \pg. are very simple. They are used when \showslides is not specified.

The \_endslides is defined as \_end primitive (preceeded by \_byehook), but slide-designer can redefine it. For example, OpTeX trick 0029 shows how to define clickable navigation to the pages and how to check the data integrity at the end of the document using \_endslides.

The \bye macro is redefined here as an alternative to \pg..
The code is somewhat more complicated when \texttt{layers} is used. Then \texttt{\_layertext} is saved to the \_\texttt{\_layertext} macro, the material before it is in \texttt{\_slidepage} box and the material after it is in \texttt{\_slidepageB} box. The pages are completed in the \texttt{\_loop} which increments the \texttt{\_layernum} register and prints page by the \texttt{\_printlayers}

\begin{Verbatim}
\texttt{\newbox\_slidepage \newbox\_slidepageB}
\texttt{\countdef\_slidelayer=1}
\texttt{\def\_slideshow}{\slidelayer=1 \_slideshowactive}
\texttt{\let\slideopen=_\_relax \_firstwins}
\texttt{\setbox\_slidepage=\vbox\_bgroupe\_bgroupe}
\texttt{\def\_slideshowactive}{\_closepage\_global\_slidelay=1 \_resetpage\_openslide}
\texttt{\def\_slideshow:}{\_closepage\_endslides}
\texttt{\def\_slideshow+:}{\_closepage\_incr\_slidelay\_decr\_pageno\_openslide}
\texttt{\let\_layers=_\_layeractive}
\texttt{\_slidelines \_to\_prevent\_hyperlink\_destds\_duplication}
\texttt{\def\_slideshow}{\_closepage\_\_resetpage\_\_openslide}
\texttt{\_newbox\_slidepage \_newbox\_slidepageB}
\texttt{\_countdef\_slidelayer=1}
\texttt{\_def\_slideshow{\_slidelayer=1 \_slideshowactive}}
\texttt{\_let\slideopen=_\_relax \_firstwins}
\texttt{\setbox\_slidepage=\vbox\_bgroupe\_bgroupe}
\texttt{\def\_slideshowactive}{\_closepage\_global\_slidelay=1 \_resetpage\_openslide}
\texttt{\def\_slideshow:}{\_closepage\_endslides}
\texttt{\def\_slideshow+:}{\_closepage\_incr\_slidelay\_decr\_pageno\_openslide}
\texttt{\let\_layers=_\_layeractive}
\texttt{\_slidelines \_to\_prevent\_hyperlink\_destds\_duplication}
\texttt{\def\_slideshow}{\_closepage\_\_resetpage\_\_openslide}
\texttt{\_newbox\_slidepage \_newbox\_slidepageB}
\texttt{\_countdef\_slidelayer=1}
\texttt{\_def\_slideshowactive}{\_closepage\_global\_slidelay=1 \_resetpage\_openslide}
\texttt{\def\_slideshow:}{\_closepage\_endslides}
\texttt{\def\_slideshow+:}{\_closepage\_incr\_slidelay\_decr\_pageno\_openslide}
\texttt{\let\_layers=_\_layeractive}
\texttt{\_slidelines \_to\_prevent\_hyperlink\_destds\_duplication}
\texttt{\def\_slideshow}{\_closepage\_\_resetpage\_\_openslide}
\texttt{\_newbox\_slidepage \_newbox\_slidepageB}
\texttt{\_countdef\_slidelayer=1}
\texttt{\_def\_slideshowactive}{\_closepage\_global\_slidelay=1 \_resetpage\_openslide}
\texttt{\def\_slideshow:}{\_closepage\_endslides}
\texttt{\def\_slideshow+:}{\_closepage\_incr\_slidelay\_decr\_pageno\_openslide}
\texttt{\let\_layers=_\_layeractive}
\texttt{\_slidelines \_to\_prevent\_hyperlink\_destds\_duplication}
\texttt{\def\_slideshow}{\_closepage\_\_resetpage\_\_openslide}
\texttt{\_newbox\_slidepage \_newbox\_slidepageB}
\texttt{\_countdef\_slidelayer=1}
\texttt{\_def\_slideshowactive}{\_closepage\_global\_slidelay=1 \_resetpage\_openslide}
\texttt{\def\_slideshow:}{\_closepage\_endslides}
\texttt{\def\_slideshow+:}{\_closepage\_incr\_slidelay\_decr\_pageno\_openslide}
\texttt{\let\_layers=_\_layeractive}
\texttt{\_slidelines \_to\_prevent\_hyperlink\_destds\_duplication}
\end{Verbatim}
When \texttt{\textbackslash slideshow} is active then the destinations of internal hyperlinks cannot be duplicated to more “virtual” pages because hyperlink destinations have to be unique in the whole document.

The \texttt{\textbackslash slideshow} creates boxes of typesetting material and copies them to more pages. So, we have to suppress creating destinations in these boxes. This is done in the \texttt{\_slidelinks} macro. We can move creating these destinations to the output routine. \texttt{\_sdestbox} is saved value of the original \texttt{\_destbox} which is redefined to do only \texttt{\addto\_destboxes\{\_sdestbox\[\langle\text{label}\rangle\}\}}. All destinations saved to \texttt{\_destboxes} are created at the start of the next output routine in the \texttt{\_pagedest} macro. The output routine removes \texttt{\_destboxes}, so each destination is created only once.

Limitations of this solution: destinations are only at the start of the page, no at the real place where \texttt{\textbackslash label} was used. The first “virtual” page where \texttt{\textbackslash label} is used includes its destination. If you want to go to the final page of the partially uncovering ideas then use \texttt{\textbackslash label\[\langle\text{label}\rangle\]\textbackslash label\texttt{\langle\text{text}\rangle}} in the last part of the page (before \texttt{\pg;}) o use \texttt{\textbackslash pgrefer} instead \texttt{\ref}.

---

The \texttt{\_settinglayer} is used in the \texttt{\_layertext} macro to prevent printing “Duplicate label” warning when it is expanded. It is done by special value of \texttt{\_slideshook} (used by the \texttt{\textbackslash label} macro). Moreover, the warning about illegal use of \texttt{\bib} in \texttt{\layers} environment is activated.

---

We must to redefine \texttt{\fnotenumpages} because the data from .ref file are less usable for implementing such a feature: the footnote should be in more layers repeatedly. But we can suppose that each page starts by \texttt{\pg;} macro, so we can reset the footnote counter by this macro.

---

2.36 Logos

---

\texttt{\_codedecl \TeX\ {\texttt{\textbackslash Text}} TeX, {	exttt{\textbackslash LuaTeX}}, etc. <2020-02-28>}} \% preloaded in format
Despite plain TeX each macro for logos ends by \ignoreslash. This macro ignores the next slash if it is present. You can use \TeX/ like this for protecting the space following the logo. This is visually more comfortable. The macros \TeX, \OpTeX, \LuaTeX, \XeTeX are defined.

The \slantcorr macro expands to the slant-correction of the current font. It is used to shifting A if the \LaTeX logo is in italic.

The expandable versions of logos used in Outlines need the expandable \ignslash (instead of the \ignoreslash).

The multilingual support
2.37 Lowercase, uppercase codes

All codes in Unicode table keep information about pairs lowercase-uppercase letters or single letter. We need to read such information and set appropriate \lccode and \uccode. The \catcode above the code 127 is not set, i.e. the \catcode=12 for all codes above 127.

The file UnicodeData.txt is read if this file exists in your TeX distribution. The format is specified at http://www.unicode.org/L2/L1999/UnicodeData.html. We read only Ll (lowercase letters), Lu (uppercase letters) and Lo (other letters) and set appropriate codes. The scanner of UnicodeData.txt is implemented here in the group (lines 6 to 15). After the group is closed then the file uni-lcuc.opm is leaved by \endinput.

If the file UnicodeData.txt does not exsists then internal data are used. They follow to the end of the file uni-lcuc.opm.
\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 \csp, 2014:
\def\tmp #1 #2 {\ifx^#1^\else
\lccode"#1="1
\ifx.#2%
\uccode"#1="1
\else
\uccode"#2="2
\lccode"#2="1
\uccode"#1="2
\fi\ea\tmp\fi}
\tmp
00AA .
00B5 039C
00BA .
00E0 00C0
00E1 00C1
00E2 00C2
00E3 00C3
00E4 00C4
00E5 00C5

...etc., 15900 similar lines (see \uni-lcuc.opm)

\subsection{Multilingual phrases and quotation marks}

Four words are generated by Op\TeX\ macros: “Chapter”, “Table”, “Figure” and “Subject”. These phrases are generated depending on the current value of the \language register, if you use \mtext{⟨phrase-id⟩}, specially \mtext{chap}, \mtext{t}, \mtext{f} or \mtext{subj}. If your macros generate more words then you can defne such words by \sdef{⟨phrase-id⟩:⟨lang-tag⟩} where ⟨phrase-id⟩ is a label for the declared word and ⟨lang-tag⟩ is a language shortcut declared by \preplang.

We can declare such language-dependent words by
\sdef{⟨phrase-id⟩:⟨lang-tag⟩}{Chapter} \sdef{⟨phrase-id⟩:cs}{Kapitola}
\sdef{⟨phrase-id⟩:en}{Table} \sdef{⟨phrase-id⟩:cs}{Table}
\sdef{⟨phrase-id⟩:en}{Table} \sdef{⟨phrase-id⟩:cs}{Table}
\sdef{⟨phrase-id⟩:en}{Table} \sdef{⟨phrase-id⟩:cs}{Table}

etc. but we use more “compact” macro \lang\⟨lang-tag⟩\ ⟨chapter⟩\ ⟨table⟩\ ⟨figure⟩\ ⟨subject⟩ for declaring them.

More phrases are auto-generated in bibliography references. They are declared by
\langb{⟨lang-tag⟩}{⟨end⟩}{⟨et-al⟩}{⟨ed⟩}{⟨cit⟩}{⟨no⟩}{⟨pp⟩}{⟨p⟩}{⟨ed⟩}{⟨eds⟩}{⟨avail-from⟩}{⟨avali-to⟩}{⟨ba-thesis⟩}{⟨ma-thesis⟩}{⟨phd-thesis⟩}. It is used similar way as the \lang above. Both these macros are used in \lang-data.opm file, see the end of section \ref{languages-opm}.

\end{document}
\today macro needs auto-generated words for each name of the month.
\_monthv \{lang-tag\} \{January\} \{February\} ... \{December\} is used for decaring them.
The language-dependent format for printing date should be declared like
\sdef{mt:today:en}{\_mttext{m\_the\_month} \_the\_day, \_the\_year}

This example declares date format for English where \{lang-tag\} is en.

\begin{verbatim}
\def \__quoteschars #1#2#3#4{
  \def \__altquotes{\__quoteschars#3#4#1#2}
  \public \__altquotes;\\
\end{verbatim}

Quotes should be tagged by "\text" and \texttt\{text\} if \{iso-code\}quotes is declared at beginning of the document (for example \enquotes). If not, then the control sequences \verb" and \verb\' are undefined. Remember, that they are used in another meaning when the \oldaccents command is used. The macros \verb" and \verb\' are not defined as \protected because we need their expansion when \outlines are created. User can declare quotes by \quoteschar\langle clqq\rangle \crqq\langle clcq\rangle \langle clcq\rangle, where \langle clcq\rangle \dots \langle clcq\rangle are normal quotes and \langle clq\rangle \dots \langle clq\rangle are alternative quotes. or use \altquotes to swap between the meaning of these two types of quotes. \enquotes, \csquotes, \frquotes, \dequotes, \skquotes are defined here. Languages in general provide the \quotes declaration macro. It declares the quotation marks depending on the actual selected language. For example, \eslang \quotes declares Spanish language including its quotation marks used for \verb"\text" and \verb\texttt\{text\} \texttt'. The language-dependent quotation marks should be declared by \quotationmarks \{lang-tag\} \{clcq\} \{crqq\} \{clcq\} \{crqq\} in the lang-data.opm file.

\begin{verbatim}
\def \__quotemacro{
  \__regmacro{\__quoteschars"#1"\__\_regquotes}
  \__regmacro{\__quoteschars\text\text''#1\__\_regquotes}
}
\end{verbatim}

The \quoteschar\langle lqq\rangle \langle rqq\rangle \langle lp\rangle \langle rp\rangle defines \verb" and \verb\' as \verb\qqA in normal mode and as expandable macros in outline mode. We want to well process the common cases: \verb"\text" or \verb\text\texttt'. 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 \verb"{\text}". This is the reason why the sub-verbatim mode runs \__qqB\langle lqq\rangle \langle rqq\rangle \langle text\rangle. The \regquotes\"\\langle L\rangle \\langle R\rangle does \def\%1\{\langle L\rangle \\langle R\rangle\} for outlines but the \" separator is active (because " and \" are active in \pdfuniddef).

\begin{verbatim}
\def \__qqB #1#2#3#4#5#6#7 #8#9{
  \__quoteschars#6#7#8#9{
    \def \__altquotes{\__quoteschars#8#9#6#7}
    \public \__altquotes;\\}
  \end{verbatim}

Sometimes should be usable to leave the markup "such" or 'such' i.e. without the first backslash. Then you can make the characters " and ' active by the \activequotes macro and leave quotes without the first backslash. First, declare \{iso-code\}quotes, then \altquotes (if needed) and finally \activequotes.
2.37.3 Languages declaration

\_preplang \langle\textit{lang-id}\rangle \langle\textit{LongName}\rangle \langle\textit{lang-tag}\rangle \langle\textit{hyph-tag}\rangle \langle\textit{lr-hyph}\rangle declares a new language. The parameters (separated by space) are

- \langle\textit{lang-id}\rangle: language identifier. It should be derived from ISO 639-1 code but additional letters can be eventually added because \langle\textit{lang-id}\rangle must be used uniquely in the whole declaration list. The \texttt{\_preplang} macro creates the language switch \langle\textit{lang-id}\rangle lang and defines also \langle\textit{lang-id}\rangle lang as a macro which expands to \langle\textit{lang-id}\rangle lang. For example, \texttt{\_preplang cs Czech ...} creates \cslang as the language switch and defines \texttt{\def\cslang{\cslang}}.
- \langle\textit{LongName}\rangle: full name of the language.
- \langle\textit{lang-tag}\rangle: language tag, which is used for setting language-dependent phrases and sorting data. If a language have two or more hyphenation patterns but a single phrases set, then we declare this language more than once with the same \langle\textit{lang-tag}\rangle but different \langle\textit{lr-hyph}\rangle.
- \langle\textit{hyph-tag}\rangle: a part of the file name where the hyphenation patterns are prepared in Unicode. The full file name is \texttt{hyph-\langle\textit{hyph-tag}\rangle.tex}. If \langle\textit{hyph-tag}\rangle is \{\} then no hyphenation patterns are loaded.
- \langle\textit{lr-hyph}\rangle: two digits, they denote \texttt{\_lefthyphenmin} and \texttt{\_righthyphenmin} values.

\_preplang allocates a new internal number by \texttt{\_newlanguage\_csname \langle\textit{lang-id}\rangle\_Patt} which will be bound to the hyphenation patterns. But the patterns nor other language data are not read at this moment. The \langle\textit{lang-id}\rangle lang is defined as \texttt{\_langinit}. When the \langle\textit{lang-id}\rangle lang switch is used firstly in a document then the language is initialized, i.e. hyphenation patterns and language-dependent data are read. The \langle\textit{lang-id}\rangle lang is re-defined itself after such initialization. \texttt{\_preplang} does also \texttt{\def\ulan{\langle\textit{longname}\rangle{\langle\textit{lang-id}\rangle}}} which is needed for the \texttt{\uselanguage} macro.

The \texttt{\_preplang} macro adds \langle\textit{lang-id}\rangle (\langle\textit{LongName}\rangle) to the \texttt{\langlist} macro which is accessible by \texttt{\langlist}. It can be used for reporting declared languages.

All languages with hyphenation patterns provided by \TeX\live are declared here. The language switches \cslang, \sklang, \delang, \pllang and many others are declared. You can declare more languages by \texttt{\_preplang} in your document, if you want. The usage of \texttt{\_preplang} with \langle\textit{lang-id}\rangle already declared is allowed. The language is re-declared in this case. This can be used in your document before first usage of the \langle\textit{lang-id}\rangle lang switch.
\_preplangmore \{lang-id\}{space}\{text\}\} declares more activities of the language switch. The \{text\} is processed whenever \_\{lang-id\}lang is invoked. If \_preplangmore is not declared for given language then \_langdefault is processed.

You can implement selecting a required script for given language, for example:

\_preplangmore ru \{\_frenchspacing\_setff{script=cyrl}\selectcyrlfont\}
\_addto\_langdefault \{_setff{}\selectlatnfont\}

The macros \selectcyrlfont and \selectlatnfont are not defined in OpTEX. If you follow this example, you have to define them after your decision what fonts will be used in your specific situation.

\_langinit \{switch\} \{lang-id\}((LongName)\{lang-tag\} [(hyph-tag)]\{lr-hyph\}). The \_langinit macro does:

- The internal language \{number\} is extracted from \_the\_\{lang-id\}Patt.
- \_def \_lan:\{number\} \{\{lang-tag\}\} for mapping from \language number to the \{lang-tag\}.
- loads hyph-\{hyph-tag\}.tex file with hyphenation patterns when \language=\{number\}.
- loads the part of lang-data.opm file with language-dependent phrases using \_langinput.
- \_def \_\{lang-id\}lang \{\_uselang\{\{lang-id\}\}\_\{lang-id\}Patt \{lr-hyph\}\}, i.e. the switch redefines itself for doing a “normal job” when the language switch is used repeatedly.
- Runs itself (i.e. \_\{lang-id\}lang) again for doing the “normal job” firstly.
The \uselang \langle lang-id \rangle \langle pre-hyph \rangle \langle post-hyph \rangle is used as “normal job” of the switch. It sets \language, \lefthyphenmin, \righthyphenmin. Finally, it runs data from \preplangmore or runs \langdefault.

The \uselanguage \langle LongName \rangle macro is defined here (for compatibility with e-plain users). Its parameter is case insensitive.

The “language data” include declarations of rules for sorting (see section 2.33), language-dependent phrases and quotation marks (see section 2.37.2). The language data are collected in the single lang-data.opm file. Appropriate parts of this file is read by \langinput \langle lang-tag \rangle. First few lines of the file looks like:

\langdata en {English} % only en, cs preloaded in format
\langdata cs {Czech} %
\langdata pl {Polish} %
\langdata de {German} %
\langdata it {Italian} %
\langdata fr {French} %

2.37.4 Data for various languages

The “language data” include declarations of rules for sorting (see section 2.33), language-dependent phrases and quotation marks (see section 2.37.2). The language data are collected in the single lang-data.opm file. Appropriate parts of this file is read by \langinput \langle lang-tag \rangle. First few lines of the file looks like:
There are analogical declaration for more languages here. Unfortunately, this file is far for completeness. I welcome you send me a part of declaration for your language.

If your language is missing in this file then a warning is reported during language initialization. You can create your private declaration in your macros (analogical as in the *lang-data.opm* file but without the \_langdata prefix). Then you will want to remove the warning about missing data. This can be done by \_nolanginput\{\langle lang-tag\rangle\} given before initialization of your language.

The whole file *lang-data.opm* is not preloaded in the format because I suppose a plenty languages here and I don’t want to waste the Ti\TeX{} memory by these declarations. Each part of this file prefixed by \_langdata\{\langle lang-tag\rangle\} is read separately when \_langinput\{\langle lang-tag\rangle\} is used. And it is used in the \_langinit macro (i.e. when the language is initialized), so the appropriate part of this file is read automatically on demand.

If the part of the *lang-data.opm* concerned by \langle lang-tag\rangle is read already then \_li:\langle lang-tag\rangle is set to R and we don’t read this part of the file again.

Data of two preferred languages are preloaded in the format:  

\begin{verbatim}
\def\langinput #1\{% \\
   \unless \_ifcsname _li:#1\_endcsname \\
   \bgroup \\
   \edef\tmp{\_noexpand\langdata #1 }\_everyeof\ea{\_tmp{}}\% \\
   \long \_ea\_def \_ea\_tmp \_ea##\_ea1\_tmp{\_readlangdata{#1}}\% \\
   \globaldefs=1 \\
   \ea\_glet \_csname _li:#1\_endcsname R\% \\
   \egroup \\
   \fi \\
\} \\
\def\_readlangdata #1\#2\{% \\
\_ifx^#2^\_opwarning{Missing data for language "#1" in lang-data.opm}\% \\
\_else \_wlog{Reading data for the language #2 (#1)}\% \\
\_fi \\
\} \\
\def\langdata #1 #2{\_endinput} \\
\def\_nolanginput #1{\_ea\_glet \_csname _li:#1\_endcsname N} \\
\public \_nolanginput ;
\end{verbatim}

2.38 Other macros

Miscellaneous macros are here.

\begin{verbatim}
\_codedecl uv {Miscenaleous <2022-05-04>} \% preloaded in format
\_useOpTeX and \_useoptex are declared as \_relax.
\_let \_useOpTeX = \_relax \_let \_useoptex = \_relax
\end{verbatim}

The \_lastpage and \_totalpages get the information from the \_currpage. The \_Xpage from *.ref* file sets the \_currpage.
We need \texttt{lv, clqq, crqq, flqq, frqq, luslang, ehyph, chyph, shyph}, for backward compatibility with \texttt{csharpplain}. Codes are set according to Unicode because we are using Czech only in Unicode when \texttt{LuaTeX} is used.

```
\_chardef\clqq=8222 \_chardef\crqq=8220
\_chardef\flqq=171 \_chardef\frqq=187
\_chardef\promile=8240
\_def\uv#1{\clqq#1\crqq}
\_let\uslang=\enlang \_let\ehyph=\enlang
\_let\chyph=\cslang \_let\shyph=\sklang
\_let\csUnicode=\csPatt \_let\czUnicode=\csPatt \_let\skUnicode=\skPatt
```

The \texttt{\letfont} was used in \texttt{csharpplain} instead of \texttt{\fontlet}.

```
\_let \letfont = \_fontlet
```

Non-breaking space in Unicode.

```
\_catcode`@=11
\_let\z@=\_zo \_let\z@skip=\_zoskip
\_newdimen\p@ \p@=1pt
\_toksdef\toks@=0
\_let\voidb@x=\_voidbox
\_chardef\@ne=1 \_chardef\tw@=2 \_chardef\thr@@=3 \_chardef\sixt@@n=16
\_mathchardef\@m=1000 \_mathchardef\@M=10000 \_mathchardef\@MM=20000
\_countdef\m@ne=22 \m@ne=-1
\_chardef\@cclv=255 \_mathchardef\@cclvi=256
\_skipdef\skip@=0
\_dimendef\dimen@=0 \_dimendef\dimen@i=1
\_dimendef\dimen@ii=2
\_countdef\count@=255
\_def\m@th{\_mathsurround\z@}
\_def\o@lign{\_lineskiplimit\z@ \_oalign}
\_def\n@space{\_nulldelimiterspace\z@ \m@th}
\_newdimen\p@renwd \p@renwd=8.75pt
\_def\alloc@#1#2#3#4#5{\_allocator#5{\_csstring#2}#3}
```

We don’t want to read \texttt{opmac.tex} unless \texttt{\input opmac} is specified.

```
\def\OPmacversion{OpTeX}
```

We allow empty lines in math formulae. It is more comfortable.

```
\_suppressmathparerror = 1
```

Lorem ipsum can be printed by \texttt{\lipsum\{range\}} or \texttt{\lorem\{range\}}, for example \texttt{\lipsum[3]} or \texttt{\lipsum[112-150]}.

First usage of \texttt{\lipsum} reads the \texttt{BIP} 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.

```
\_lipsumdot
```

LuaTeX version 1.14 and newer provides \partokenname which allows to specify something different than \par at empty lines. We set \par (see below) in OpteX version 1.04+ and newer. Some macros were rewritten due to this change. And we copy old versions of these changed macros here in order to allow to use older LuaTeX versions where \partokenname is not provided.

Note that your macros where a parameter is separated by the empty line must be changed too. Use \def\macro #1\par{...} instead \def\macro #1\par{...}.

We set \partokenname to \par in order to keep the name \par in the public namespace for end users. I.e. a user can say \def\par{paragraph} for example without crash of processing the document. So
Moreover, we set \partokencontext to one in order to the \_par token is inserted not only at empty lines, but also at the end of \vbox, \vtop and \vcenter if horizontal mode is opened here. This differs from default \TeX behavior where horizontal mode is closed in these cases without inserting par token. We set \_partokenset to defined value 1 in order to the macro programmer can easily check these settings in Op\TeX format by \ifx\_partokenset\undefined ... \else ...\fi.

```
\_partokenname\_par
\_partokencontext=1
\_let\_partokenset=1
\fi
```

## 2.39 Lua code embedded to the format

The file \texttt{optex.lua} is loaded into the format in \texttt{optex.ini} as byte-code and initialized by \texttt{\everyjob}, see section 2.1.

The file implements part of the functionality from \texttt{luatexbase} namespace, nowadays defined by \texttt{\LaTeX} kernel. \texttt{luatexbase} deals with modules, allocators, and callback management. Callback management is a nice extension and is actually used in Op\TeX. Other functions are defined more or less just to suit luatolfload’s use.

The allocations are declared in subsection 2.39.2, callbacks are implemented in subsection 2.39.3 and handling with colors can be found in the subsection 2.39.5.

```
local fmt = string.format

local function err(message)
  error("\nerror: \"..message..\"\n")
end

local function registernumber(name)
  return token.create(name).index
end

local function mdfive(file)
  local fh = io.open(file, "rb")
  if fh then
    local data = fh:read("a")
    fh:close()
    tex.print(md5.sumhexa(data))
  end
end
```

### 2.39.1 General

Define namespace where some Op\TeX functions will be added.

```
local optex = \_ENV.optex or {}
\_ENV.optex = optex
```

Error function used by following functions for critical errors.

```
local function registernumber(name)
  return token.create(name).index
end

\_ENV.registernumber = registernumber
```

For a \chardef, \countdef, etc., csname return corresponding register number. The responsibility of providing a \texttt{\XXdef} name is on the caller.

```
local function mdfive(file)
  local fh = io.open(file, "rb")
  if fh then
    local data = fh:read("a")
    fh:close()
    tex.print(md5.sumhexa(data))
  end
end
```

### 2.39.2 Allocators

```
local alloc = \_ENV.alloc or {}
\_ENV.alloc = alloc
```

MD5 hash of given file.
An attribute allocator in Lua that cooperates with normal \TeX\ allocator.

```lua
local attributes = {}
function alloc.new_attribute(name)
    local cnt = tex.count("_attributealloc") + 1
    if cnt > 65534 then
        tex.error("No room for a new attribute")
    else
        tex.setcount("global", "_attributealloc", cnt)
        texio.write_nl("log", "'"..name.."'=\attribute..tostring(cnt))
        attributes[name] = cnt
        return cnt
    end
end
```

Allocator for Lua functions ("pseudoprimitives"). It passes variadic arguments ("...") like "global" to \texttt{token.set\_lua}.

```lua
local function_table = lua.get_functions_table()
local function define_lua_command(csname, fn, ...)
    local luafnalloc = #function_table + 1
    token.set_lua(csname, luafnalloc, ...) -- WARNING: needs \TeX\ 1.08 (2019) or newer
    function_table[luafnalloc] = fn
end
_ENV.define_lua_command = define_lua_command
optex.define_lua_command = define_lua_command
```

### 2.39.3 Callbacks

```lua
local callback = _ENV.callback or {}
_ENV.callback = callback
```

Save \texttt{callback.register} function for internal use.

```lua
local callback_register = callback.register
function callback.register(name, fn)
    err("direct registering of callbacks is forbidden, use 'callback.add\_to\_callback'")
end
```

Table with lists of functions for different callbacks.

```lua
local callback_functions = {}
Table that maps callback name to a list of descriptions of its added functions. The order corresponds with \texttt{callback\_functions}.

```lua
local callback_description = {}
Table used to differentiate user callbacks from standard callbacks. Contains user callbacks as keys.

```lua
local user_callbacks = {}
Table containing default functions for callbacks, which are called if either a user created callback is defined, but doesn’t have added functions or for standard callbacks that are “extended” (see \texttt{mlist\_to\_hlist} and its pre/post filters below).

```lua
local default_functions = {}
Table that maps standard (and later user) callback names to their types.

```lua
local callback_types = {
    -- file discovery
    find_read_file = "exclusive",
    find_write_file = "exclusive",
    find_font_file = "data",
    find_output_file = "data",
    find_format_file = "data",
    find_vf_file = "data",
    find_map_file = "data",
    find_enc_file = "data",
    find_pk_file = "data",
    find_data_file = "data",
}
```

204
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
Return a list containing descriptions of added callback functions for specific callback.

```lua
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.

```lua
function callback.create_callback(name, cbtype, default)
    callback_types[name] = cbtype
    default_functions[name] = default or nil
    user_callbacks[name] = true
end
```

Add a function to the list of functions executed when callback is called. For standard luatex callback a proxy function that calls our machinery is registered as the real callback function. This doesn’t happen for user callbacks, that are called manually by user using call_callback or for standard callbacks that have default functions – like mlist_to_hlist (see below).

```lua
local call_callback
function callback.add_to_callback(name, fn, description)
    if user_callbacks[name] or callback_functions[name] or default_functions[name] then
        -- either:
        -- a) user callback - no need to register anything
        -- b) standard callback that has already been registered
        -- c) standard callback with default function registered separately
        -- (mlist_to_hlist)
    elseif callback_types[name] then
        -- This is a standard luatex callback with first function being added,
        -- register a proxy function as a real callback. Assert, so we know
        -- when things break, like when callbacks get redefined by future
        -- luatex.
        callback_register(name, function(...) return call_callback(name, ...) end)
    else
        err("cannot add to callback ".name.." - no such callback exists")
    end
end
```

Remove a function from the list of functions executed when callback is called. If last function in the list is removed delete the list entirely.
function callback.remove_from_callback(name, description)
    local descriptions = callback_description[name]
    local index
    for i, desc in ipairs(descriptions) do
        if desc == description then
            index = i
            break
        end
    end
    table.remove(descriptions, index)
    local fn = table.remove(callback_functions[name], index)
    if #descriptions == 0 then
        -- Delete the list entirely to allow easy checking of "truthiness".
        callback_functions[name] = nil
    end
    if not user_callbacks[name] and not default_functions[name] then
        -- this is a standard callback with no added functions and no
        -- default function (i.e. not mlist_to_hlist), restore standard
        -- behaviour by unregistering.
        callback_register(name, nil)
    end
    return fn, description
end

local function reverse ipairs(t)
    local i, n = #t + 1, 1
    return function()
        i = i - 1
        if i >= n then
            return i, t[i]
        end
    end
end

Call all functions added to callback. This function handles standard callbacks as well as user created
callbacks. It can happen that this function is called when no functions were added to callback – like for
user created callbacks or mlist_to_hlist (see below), these are handled either by a default function (like
for mlist_to_hlist and those user created callbacks that set a default function) or by doing nothing for
empty function list.

function callback.call_callback(name, ...
    local cbtype = callback_types[name]
    -- either take added functions or the default function if there is one
    local functions = callback_functions[name] or {default_functions[name]}
    if cbtype == nil then
        err("cannot call callback \"..name..\" - no such callback exists")
    elseif cbtype == "exclusive" then
        -- only one function, atleast default function is guaranteed by
        -- create_callback
        return functions[1](...)
    elseif cbtype == "simple" then
        -- call all functions one after another, no passing of data
        for _, fn in ipairs(functions) do
            fn(...)
        end
        return
    elseif cbtype == "data" then
        -- pass data (first argument) from one function to other, while keeping
        -- other arguments
        local data = (...)
        for _, fn in ipairs(functions) do
            data = fn(data, select(2, ...))
        end
        return data
    end
end

helper iterator generator for iterating over reverselist callback functions
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
call_callback = callback.call_callback
Create "virtual" callbacks pre/post_mlist_to_hlist_filter by setting mlist_to_hlist callback. The default behaviour of mlist_to_hlist is kept by using a default function, but it can still be overridden by using add_to_callback.

default_functions["mlist_to_hlist"] = node.mlist_to_hlist
callback.create_callback("pre_mlist_to_hlist_filter", "list")
callback.create_callback("post_mlist_to_hlist_filter", "reverselist")
callback_register("mlist_to_hlist", function(head, ...)
  -- pre_mlist_to_hlist_filter
  local new_head = call_callback("pre_mlist_to_hlist_filter", head, ...)
  if new_head == false then
    node.flush_list(head)
    return nil
  elseif new_head ~= true then
    head = new_head
  end
  -- mlist_to_hlist means either added functions or standard luatex behavior
  -- of node.mlist_to_hlist (handled by default function)
  head = call_callback("mlist_to_hlist", head, ...)
  -- post_mlist_to_hlist_filter
  new_head = call_callback("post_mlist_to_hlist_filter", head, ...)
  if new_head == false then
    node.flush_list(head)
    return nil
  elseif new_head ~= true then
    head = new_head
  end
  return head
end)
callback.create_callback("pre_shipout_filter", "list")
callback.create_callback("pre_shipout_filter", "list")
tex_setbox = tex.setbox
For preprocessing boxes just before shipout we define custom callback. This is used for coloring based on attributes. There is however a challenge - how to call this callback? We could redefine \shipout and \pdfform (which both run ship_out procedure internally), but they would lose their primitive meaning – i.e. \immediate wouldn’t work with \pdfform. The compromise is to require anyone to run \_preshipout⟨destination box number⟩⟨box specification⟩ just before \shipout or \pdfform if they want to call pre_shipout_filter (and achieve colors and possibly more).
local token_scanint = token.scan_int
local token_scanlist = token.scan_list
define_lua_command("_preshipout", function()
  local boxnum = token_scanint()
  local head = token_scanlist()
  head = call_callback("pre_shipout_filter", head)
tex_setbox(boxnum, head)
end)

Compatibility with \LaTeX{} through luatexbase namespace. Needed for luaotfload.

_ENV.luatexbase = {
  registernumber = registernumber,
  attributes = attributes,
  -- `provides_module` is needed by older version of luaotfload
  provides_module = function() end,
  new_attribute = alloc.new_attribute,
  callback_descriptions = callback.callback_descriptions,
  create_callback = callback.create_callback,
  add_to_callback = callback.add_to_callback,
  remove_from_callback = callback.remove_from_callback,
  call_callback = callback.call_callback,
  callbacktypes = {},
}

\tracingmacros callback registered. Use \tracingmacros=3 or \tracingmacros=4 if you want to see the result.

callback.add_to_callback("input_level_string", function(n)
  if tex.tracingmacros > 3 then
    return \[n\]
  elseif tex.tracingmacros > 2 then
    return ~..\string.rep(\.,n)
  else
    return ""
  end
end, "._tracingmacros")

2.39.4 Management of PDF page resources

Traditionally, pdf\LaTeX{} allowed managing PDF page resources (graphics states, patterns, shadings, etc.) using a single toks register, \texttt{\pdffileresources}. This is insufficient due to the expected PDF object structurer and also because many “packages” want to add page resources and thus fight for the access to that register. We add a finer alternative, which allows adding different kinds of resources to a global page resources dictionary. Note that some resource types (fonts and XObjects) are already managed by Lua\LaTeX{} and shouldn’t be added!

XObject forms can also use resources, but there are several ways to make Lua\LaTeX{} reference resources from forms. It is hence left up to the user to insert page resources managed by us, if they need them. For that, use \texttt{pdf.get_page_resources()}, or the below \TeX{} alternative for that.

local pdfdict_mt = {
  __tostring = function(dict)
    local out = {\text{"<<"}
    for k, v in pairs(dict) do
      out[#out+1] = fmt(\/%s \%s", tostring(k), tostring(v))
    end
    out[#out+1] = \text{">>"}
    return table.concat(out, \text{"\n"})
  end,
}
local function pdf_dict(t)
  return setmetatable(t or {}, pdfdict_mt)
end
optex.pdf_dict = pdf_dict

local pdfdict_objects = {}
local page_resources = ()
function pdf.add_page_resource(type, name, value)
local resources = page_resources[type]
if not resources then
  local obj = pdf.reserveobj()
pdf.setpageresources(fmt("%s /%s %d 0 R", pdf.get_page_resources(), type, obj))
  resource_dict_objects[type] = obj
  resources = pdf_dict()
  page_resources[type] = resources
end
page_resources[type][name] = value
end

function pdf.get_page_resources()
  return pdf.getpageresources() or"
end

New “pseudo” primitives are introduced. \_addpageresource{⟨type⟩}{⟨PDF name⟩}{⟨PDF dict⟩}
adds more resources of given resource ⟨type⟩ to our data structure. \_pageresources expands to the saved ⟨type⟩s and object numbers.

define_lua_command("_addpageresource", function()
  pdf.add_page_resource(token.scan_string(), token.scan_string(), token.scan_string())
end)
define_lua_command("_pageresources", function()
tex.print(pdf.get_page_resources())
end)

We write the objects with resources to the PDF file in the finish_pdffile callback.

callback.add_to_callback("finish_pdffile", function()
  for type, dict in pairs(page_resources) do
    local obj = resource_dict_objects[type]
    pdf.immediateobj(obj, tostring(dict))
  end
end)

2.39.5 Handling of colors and transparency using attributes

Because LuaTeX doesn’t do anything with attributes, we have to add meaning to them. We do this by intercepting TeX just before it ships out a page and inject PDF literals according to attributes.

local node_id = node.id
local node_subtype = node.subtype
local glyph_id = node_id("glyph")
local rule_id = node_id("rule")
local glue_id = node_id("glue")
local hlist_id = node_id("hlist")
local vlist_id = node_id("vlist")
local disc_id = node_id("disc")
local whatsit_id = node_id("whatsit")
local pdfliteral_id = node_subtype("pdf_literal")
local pdfsave_id = node_subtype("pdf_save")
local pdfrestore_id = node_subtype("pdf_restore")
local token_getmacro = token.get_macro
local direct = node.direct
local todirect = direct.todirect
local tonode = direct.tonode
local getfield = direct.getfield
local setfield = direct.setfield
local getwhd = direct.getwhd
local getid = direct.getid
local getlist = direct.getlist
local setlist = direct.setlist
local getleader = direct.getleader
local getattribute = direct.get_attribute
local insertbefore = direct.insert_before
local copy = direct.copy
local traverse = direct.traverse
local one_bp = tex.sp("1bp")

The attribute for coloring is allocated in colors.opm
Now we define function which creates whatsit nodes with PDF literals. We do this by creating a base literal, which we then copy and customize.

```.lua
local pdf_base_literal = direct.new("whatsit", "pdf_literal")
setfield(pdf_base_literal, "mode", 2) -- direct mode
local function pdfliteral(str)
    local literal = copy(pdf_base_literal)
    setfield(literal, "data", str)
    return literal
end
optex.directpdfliteral = pdfliteral
```

The function `colorize(head, current, current_stroke, current_tr)` goes through a node list and injects PDF literals according to attributes. Its arguments are the head of the list to be colored and the current color for fills and strokes and the current trasparency attribute. It is a recursive function – nested horizontal and vertical lists are handled in the same way. Only the attributes of “content” nodes (glyphs, rules, etc.) matter. Users drawing with PDF literals have to set color themselves.

Whatsit node with color setting PDF literal is injected only when a different color or transparency is needed. Our injection does not care about boxing levels, but this isn’t a problem, since PDF literal whatsits just instruct the \shipout related procedures to emit the literal.

We also set the stroke and non-stroke colors separately. This is because stroke color is not always needed – LuaTEX itself only uses it for rules whose one dimension is less than or equal to 1 bp and for fonts whose mode is set to 1 (outline) or 2 (outline and fill). Catching these cases is a little bit involved. For example rules are problematic, because at this point their dimensions can still be running ($-2^{30}$) – they may or may not be below the one big point limit. Also the text direction is involved. Because of the negative value for running dimensions the simplistic check, while not fully correct, should produce the right results. We currently don’t check for the font mode at all.

Leaders (represented by glue nodes with leader field) are not handled fully. They are problematic, because their content is repeated more times and it would have to be ensured that the coloring would be right even for e.g. leaders that start and end on a different color. We came to conclusion that this is not worth, hence leaders are handled just opaquely and only the attribute of the glue node itself is checked. For setting different colors inside leaders, raw PDF literals have to be used.

We use the `node.direct` way of working with nodes. This is less safe, and certainly not idiomatic Lua, but faster and codewise more close to the way \TeX works with nodes.
list, current, current_stroke, current_tr =
colorize(list, current, current_stroke, current_tr)
setlist(n, list)
elseif id == disc_id then
  -- at this point only no-break (replace) list is of any interest
  local replace = getfield(n, "replace")
  if replace then
    replace, current, current_stroke, current_tr =
colorize(replace, current, current_stroke, current_tr)
  setfield(n, "replace", replace)
end
else
  local fill_needed, stroke_needed = is_color_needed(head, n, id, subtype)
  local new = getattribute(n, color_attribute) or 0
  local newtr = getattribute(n, transp_attribute) or 0
  local newliteral = nil
  if current ~= new and fill_needed then
    newliteral = token_getmacro("_color:"..new)
    current = new
  end
  if current_stroke ~= new and stroke_needed then
    stroke_color = token_getmacro("_color-s:"..current)
    if stroke_color then
      if newliteral then
        newliteral = fmt("%s %s", newliteral, stroke_color)
      else
        newliteral = stroke_color
      end
    current_stroke = new
  end
  if newtr ~= current_tr and fill_needed then -- (fill_ or stroke_needed) = fill_neded
    if newliteral ~= nil then
      newliteral = fmt("%s /tr%d gs", newliteral, stroke_color)
    else
      newliteral = fmt("/tr%d gs", newtr)
    end
    current_tr = newtr
  end
  if newliteral then
    head = insertbefore(head, n, pdfliteral(newliteral))
  end
end
return head, current, current_stroke, current_tr
end

Colorization should be run just before shipout. We use our custom callback for this. See the definition of pre_shipout_filter for details on limitations.

callback.add_to_callback("pre_shipout_filter", function(list)
-- By setting initial color to -1 we force initial setting of color on
every page. This is useful for transparently supporting other default
-- colors than black (although it has a price for each normal document).
local list = colorize(todirect(list), -1, -1, 0)
return tonode(list)
end, "_colors")

We also hook into luaotfload’s handling of color and transparency. Instead of the default behavior (inserting colorstack whatsis) we set our own attribute. On top of that, we take care of transparency resources ourselves.

The hook has to be registered after luaotfload is loaded.

local setattribute = direct.set_attribute
local token_setmacro = token.set_macro
local color_count = registernumber("_colorcnt")
local tex_getcount, tex_setcount = tex.getcount, tex.setcount

local function set_node_color(n, color) -- "1 0 0 rg" or "0 g", etc.
local attr = tonumber(token_getmacro("_color::"..color))
if not attr then
    attr = tex_getcount(color_count)
    tex_setcount(color_count, attr + 1)
    local strattr = tostring(attr)
    token_setmacro("_color::"..color, strattr)
    token_setmacro("_color:"..strattr, color)
    token_setmacro("_color-s:"..strattr, string.upper(color))
end
setattribute(todirect(n), color_attribute, attr)

function optex.hook_into_luaotfload()
    -- color support for luaotfload v3.13+, otherwise broken
    pcall(luaotfload.set_colorhandler, function(head, n, rgbcolor) -- rgbcolor = "1 0 0 rg"
        set_node_color(n, rgbcolor)
        return head, n
    end)
    -- transparency support for luaotfload v3.22+, otherwise broken
    pcall(function()
        luatexbase.add_to_callback("luaotfload.parse_transparent", function(input) -- from "00" to "FF"
            -- in luaotfload: 0 = transparent, 255 = opaque
            -- in optex: 0 = opaque, 255 = transparent
            local alpha = tonumber(input, 16)
            if not alpha then
                tex.error("Invalid transparency specification passed to font")
                return nil
            elseif alpha == 255 then
                return nil -- this allows luaotfload to skip calling us for opaque style
            end
            local transp = 255 - alpha
            local transpv = fmt("%.3f", alpha / 255)
            pdf.add_page_resource("ExtGState", fmt("tr%d", transp), pdf_dict{ca = transpv, CA = transpv})
            pdf.add_page_resource("ExtGState", "tr0", pdf_dict{ca = 1, CA = 1})
            return transp -- will be passed to the below function
        end, "optex")
    end)
    luaotfload.set_transparenthandler(function(head, n, transp)
        setattribute(n, transp_attribute, transp)
        return head, n
    end)
end

2.40 Printing documentation

The \printdoc (filename)(space) and \printdoc{filename}(space) commands are defined after the file doc.opm is load by \load [doc].

The \printdoc{filename} starts reading of given (filename) from the second line. The file is read in the listing mode. The \printdoc{filename} starts reading given (filename) from the first occurrence of the \endcode. The file is read in normal mode (like \input {filename}).

The listing mode prints the lines as a listing of a code. This mode is finished when first \endcode occurs or first \_doc occurs. At least two spaces or one tab character must precede before such \_doc. On the other hand, the \endcode must be at the left edge of the line without spaces. If this rule is not met then the listing mode continues.
If the first line or the last line of the listing mode is empty then such lines are not printed. The maximal number of printed lines in the listing mode is \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} ... \texttt{\endtt} pair is shifted by three letters left. The reason is that the three spaces of indentation is recommended in the \texttt{\_doc} ... \texttt{\_cod} pair and this shifting is compensation for this indentation.

The \texttt{\_cod} macro ignores the rest of the current line and starts the listing mode again.

When the listing mode is finished by the \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 \texttt{\_endcode}) in the listing mode.

The main documentation point is denoted by \texttt{\langle\sequence\rangle} in red, for example \texttt{\foo}. The user documentation point is the first occurrence of \texttt{\langle\sequence\rangle}, 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 precedes. Finally, the \texttt{\langle\sequence\rangle} (for example \texttt{\foo}) are hyperlinks to the user documentation point.

By default, the hyperlink from main documentation point to the user documentation point is active only if it is backward link, i.e. the main documentation point is given later. The reason is that we don’t know if such user documentation point will exist when creating main documentation point and we don’t want broken links. If you are sure that user documentation point will follow then use prefix \texttt{\fw} before \texttt{\}, for example \texttt{\fw\foo} is main documentation point where the user documentation point is given later and forward hyperlink is created here.

Control sequences and their page positions of main documentation points and user documentation points are saved to the index.

The listing mode creates all control sequences which are listed in the index as an active link to the main documentation point of such control sequence and prints them in blue. Moreover, active links are control sequences of the type \texttt{\_foo} or \texttt{.foo} although the documentation mentions only \texttt{foo}. Another text is printed in black.

The listing mode is able to generate external links to another OpTEX-like documentation, if the macros \texttt{\langle\csname\rangle} and \texttt{\el:\langle\csname\rangle} are defined. The second macro should create a hyperlink using \texttt{\_tmpa} where the link name of the \texttt{\csname} is saved and \texttt{\_tmpb} where the name of the \texttt{\csname} to be printed is saved (\texttt{\tmpb} can include preceding \_ or \. unlike \texttt{\_tmpa}). For example, suppose, that we have created \texttt{optex-doc.eref} file by:

```
TEXINPUTS='.;$TEXMF/{doc,tex}//' optex optex-doc
grep Xindex optex-doc.ref > optex-doc.eref
```

The \texttt{.eref} file includes only \texttt{\Xindex{\langle\csname\rangle}{}} lines from \texttt{optex-doc.ref} file. Then we can use following macros:

```
\def\Xindex#1#2{\def{,#1}{}\slet{el:#1}{optexdoclink}\
\def\optexdoclink{%
  \edef\extlink{url:\optexdocurl\csstring\#cs:\_tmpa}\%
  \_ea\urlactive\_ea[extlink]{\Cyan}{\csstring{\_tmpb}}%}
\def\optexdocurl{http://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf}
\isfile{optex-doc.eref}\iftrue \input{optex-doc.eref}\fi
```

All \texttt{\el:\langle\csname\rangle}, where \texttt{\csname} is from \texttt{optex-doc.ref}, have the same meaning: \texttt{\optexdoclink} in this example. And \texttt{\optexdoclink} creates the external link in \texttt{\Cyan} color.

2.40.1 Implementation

General decalarations.
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 \catcode \=11 sets dot as letter in listngs (for package documentation where \.foo sequences exist).

The scanner of the control sequences in the listing mode replaces all occurrences of \ by \makecs. This macro reads next tokens and accumulates them to \tmpa as long as they have category 11. It means that \tmpa includes the name of the following control sequence when \makecsF is run. The printing form of the control sequence is set to \tmpb and the test of existence, ⟨csname⟩ is performed. If it is true then active hyperlink is created. If not, then the first _ or . is removed from \tmpa and the test is repeated.
By default the internal link is created by \_intlink inside listing mode. But you can define \el:
⟨\csname⟩⟩ which has precedence and it can create an external link. The \_tmpa includes the name used in the link and \_tmpb is the name to be printed. See \_makecsF above and the example at the beginning of this section.

\_def\_intlink{\_link[cs:\_tmpa]{\ulinkcolor}{\csstring\_tmpb}}

The lines in the listing mode have a yellow background.

\docfile is currently documented file.
\printdoc and \printdoctail macros are defined here.

You can do \verbinput \vitt{\langle filename⟩} \langle (from)--(to) ⟩ \langle filename⟩ if you need analogical design like in listing mode.
The Index entries are without the trailing backslash in .ref file. When printing Index, we distinguish the Index entries with their main documentation point (they are created as links and backslash is added). Index entries with only user documentation points have backslash added but no link is created. Other index entries are printed as usual without backslash.

If this macro is loaded by \load then we need to initialize catcodes using the \afterload macro.

The \texttt{<something>} will be print as \texttt{⟨something⟩}.

Main documentation points and hyperlinks to/from it. Main documentation point: \texttt{\`foo}. User documentation point: \texttt{\^\`foo}, first occurrence only. The next occurrences are only links to the main documentation point. Link to user documentation point: \texttt{~\`foo}.
The \texttt{fw} macro for forward links to user documentation point (given later) is defined here.

\begin{verbatim}
\def\fw\string{\slet{cs:\csstring{#1}}{\string#1}}
\public \fw ;
\end{verbatim}
Index

There are all control sequences used in OpTeX except \TeX primitives. If you want to know something about \TeX primitives then you can use another index from \TeX in a Nutshell.

\_abeliskip 136
\_abelivetiskp 131, 133
\activequotes 195
\addcitelist 161
\addcolor 119
\addextgstate 117, 150
\adictcorr 110
\addpageresource 150, 210
\address 25, 188–189
\addtabitemx 155
\addto 28, 39, 111
\addtomodlist 80
\addumathfont 96–97, 99
\aldef 17, 28, 39
\alot 91
\advancepageno 111, 113
\afterfi 28, 42
\afteritcorr 110
\afterload 53
\aheadto 28, 39, 99
\allocator 40
\allowbreak 59
\altquotes 195
\ascisortingtrue 183
\athe 137
\authlist 176
\authorname 165
\b 61
\backgroundbox 112
\backgroundpic 148
\balancecolumns 158
\banner 39
\basefilename 28, 36
\bbchar 85, 102
\bbdigits 101
\bbig 90
\bbigl 90
\bbigm 90
\bbiggm 90
\bbiggr 90
\bbigl 90
\bbiglq 90
\bbig 90
\_bigbreak 59
\_bigg 90
\_biggl 90
\_biggl 90
\_biggm 90
\_biggr 90
\_bigl 90
\_biglq 90
\_big 90
\_bfgreek 101, 103
\_bfGreek 101
\_bfraikvariables 101
\_bfvariables 101
\_bg 8–9, 68–69, 77, 85, 102
\_bfgreek 101
\_bfgreek 101
\_bfgreek 101
\_bfrakvariables 101
\_bfvariables 101
\_bg 8–9, 68–69, 77, 85, 102
\_bib 20–21, 27, 163
\_bibA 163
\_bibB 163
\_bibg 163
\_bibm 163
\_bibmark 160, 163, 165
\_bibnum 124, 160
\_biboptions 50, 169
\_bip 160
\_bibpart 21, 50, 160
\_bisskip 163
\_biptexhook 50, 164
\_biwarning 164, 168
\_big 90
\_Big 90
\_bigbreak 59
\_bigg 90
\_Bigg 90
\_biggl 90
\_Biggl 90
\_biggm 90
\_Biggm 90
\_biggr 90
\_Biggr 90
\_bigl 90
\_Bigl 90
\_bigr 90
\_Bigr 90
\_bgreek 101
\_bgreek 101
\_big 90
\_Big 90
\_bfrakvariables 101
\_bfvariables 101
\_bg 8–9, 68–69, 77, 85, 102
\_bgskip 59
\_bgskipamount 47
\_bisangreek 101
\_bisansGreek 101
\_bisansvariables 101
\_bivariables 101
\_bigr 90
\_bigl 90
\_blue 21, 116
\_bold 92
\_boldify 71, 110
\_boldmath 9, 84, 86, 96–99, 109
\_boldunimath 98
\_bordermatrix 93
\_bordermatrixwithdelims 93
\_boxlines 188
\_bp 28, 55
\_bp 56
\_bprinta 164, 167
\_bprintb 164, 167
\_bprintrc 164, 167
\_bprintrv 165, 167
\_bracedparam 54
\_break 59
\_Brown 116
\_bsangdigits 101
\_bsanggreek 101
\_bsangGreek 101
\_bsansvariables 101
\_bslash 38
\_buildrel 92
\_bye 33, 39, 62
\_byehook 39, 121
\_c 61
\_cal 85, 102
\_calvariables 101
\_caption 10–12, 27, 135
\_captionformat 135
\_captionsep 135
\_cases 93
\_casesof 28, 45–46
\_catalogexclude 82
\_catalogmathsample 82
\_catalogonly 82
\_catalogsample 82
\_catcode 55
\_cdots 91
\_centerline 60
\_chap 10, 12, 17–18, 27, 54, 131, 133
\_chapfont 71, 131
\_chapx 132
\_checkexists 35
\_chyp 24, 201
\_circle 148–149
\_circleparams 52
\_cite 12, 20–21, 27, 160, 164
\_citeA 161
\_citeborder 12, 124
\_cite1 161
\_classfam 100
\_clipcircle 24, 151
\_clipinpath 150
\_clipincircle 24, 151
\_clipinpah 150
\_clq 201
\_cmkcolordef 119