OpTEX

Format Based on Plain TEX and OPmac[1]

Version 1.18

Petr Olšák, 2020, 2021, 2022, 2023, 2024, 2025

http://petr.olsak.net/optex

OpTEX is LuaTEX format with Plain TEX and OPmac. Only LuaTEX engine is supported (more exactly: LuaHBTEX engine is used from 2025).

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 `.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 (`\foo` versus `\_foo`). The namespaces for macro writers are designed too.

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

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

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

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

---

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

[2] All these features are implemented by TEX macros, no external program is needed.

Contents

Chapter 1

User documentation

1.1 Starting with OpTeX

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

```
optex document
```

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

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

A minimal document should be

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

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

A somewhat larger example with common settings should be:

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

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

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

1.2 Page layout

1.2.1 Setting the margins

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

```
\margins/⟨pg⟩ ⟨fmt⟩ (⟨left⟩,⟨right⟩,⟨top⟩,⟨bot⟩)⟨unit⟩
  example:
\margins/1 a4 (2.5,2.5,2,2)cm
```

Parameters are:

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

---

[1] This is a technical limitation of LuaTeX for fonts downloaded in formats: only 8bit fonts can be preloaded.

Each of the parameters ⟨*left*⟩, ⟨*right*⟩, ⟨*top*⟩, ⟨*bot*⟩ can be empty. If both ⟨*left*⟩ and ⟨*right*⟩ are nonempty then `\hsize` is set. Else `\hsize` is unchanged. If both ⟨*left*⟩ and ⟨*right*⟩ are empty then typesetting area is centered in the paper format. The analogical rule works when ⟨*top*⟩ or ⟨*bot*⟩ parameter is empty (`\vsize` instead `\hsize` is used). Examples:

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

If ⟨*pg*⟩`=1` then all pages have the same margins. If ⟨*pg*⟩`=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 ⟨*left*⟩ is replaced by ⟨*right*⟩ and vice versa.

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

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

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

```
\margins/1 (100,200) (7,7,7,7)mm
```

declares the paper 100×200 mm with all four margins 7 mm. The spaces before and after ⟨*fmt*⟩ parameter are necessary.

The command `\magscale[`⟨*factor*⟩`]` scales the whole typesetting area. The fixed point of such scaling is the upper left corner of the paper sheet. Typesetting (breakpoints etc.) is unchanged. All units are relative after such scaling. Only paper format's dimensions stay unscaled. Example:

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

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

1.2.2 Concept of the default page

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

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

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

```
\headline = {}
\footline = {\_hss\_rmfixed \_folio \_hss} % \folio expands to page number
\headlinedist = 14pt % from baseline of \headline to top of page body
\footlinedist = 24pt % from last line in pagebody to baseline of footline
```

The page body should be divided into top insertions (floating tables and figures) followed by a real text and followed by footnotes. Typically, the only real text is here.

The `\pgbackground` tokens list is empty by default but it can be used for creating a background of each page (colors, picture, watermark for example). The macro `\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 `\footnote` can be used as usual. But a new macro `\fnote{`⟨*text*⟩`}` is defined. The footnote mark is added automatically and it is numbered on each chapter from one[2]. The ⟨*text*⟩ is scaled to 80 %. User can redefine footnote mark or scaling, as shown in the section 2.34.

The `\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 `\fnotemark`⟨*numeric-label*⟩ inside the box: only the footnote mark is generated here. When the box is finished you can use `\fnotetext{`⟨*text*⟩`}`. This macro puts the ⟨*text*⟩ to the footnote. The ⟨*numeric-label*⟩ has to be `1` if only one such command is in the box. Second `\fnotemark` inside the same box has to have the parameter `2` etc. The same number of `\fnotetext`s have to be written after the box as the number of `\fnotemark`s inserted inside the box. Example:

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

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

The ⟨*text*⟩ is formatted as a little paragraph with the maximal width `\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 `\mnote` in the text. The exceptions are possible by using the `up` keyword: `\mnote up`⟨*dimen*⟩`{`⟨*text*⟩`}`. You can set such ⟨*dimen*⟩ to each `\mnote` manually in final printing in order to margin notes do not overlap. The positive value of ⟨*dimen*⟩ shifts the note up and negative value shifts it down. For example `\mnote up 2\baselineskip{`⟨*text*⟩`}` shifts this marginal note two lines up.

1.3 Fonts

1.3.1 Font families

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

---

[2] You can declare `\fnotenumglobal` if you want footnotes numbered in whole document from one or `\fnotenumpages` if you want footnotes numbered at each page from one. Default setting is `\fnotenumchapters`

If you write `\fontfam[?]` then all font families registered in OpTEX are listed on the terminal and in the log file. If you write `\fontfam[catalog]` then a catalog of all fonts registered in OpTEX and available in your TEX system is printed. See also this catalog.

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

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

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

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

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

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

1.3.2 Font sizes

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

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

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

First usage of `\typosize` or `\typoscale` macro in your document sets so-called *main values*, i. e. main font size and main baselineskip. They are internally saved in registers `\mainfosize` and `\mainbaselineskip`.

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

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

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

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

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

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

1.3.3 Typesetting math

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

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

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

The following math alphabets are available:

`\mit`	`% mathematical variables`	$abc{-}xyz, ABC{-}XYZ$
`\it`	`% text italics`	*abc–xyz, ABC–XYZ*
`\rm`	`% text roman`	abc–xyz, ABC–XYZ
`\cal`	`% normal calligraphics`	$\mathcal{ABC}{-}\mathcal{XYZ}$
`\script`	`% script`	$\mathscr{ABC}{-}\mathscr{XYZ}$
`\frak`	`% fracture`	$\mathfrak{abc}{-}\mathfrak{xyz}, \mathfrak{ABC}{-}\mathfrak{XYZ}$
`\bbchar`	`% double stroked letters`	$\mathbb{ABC}{-}\mathbb{XYZ}$
`\bf`	`% sans serif bold`	$\mathbf{\mathsf{abc}}{-}\mathbf{\mathsf{xyz}}, \mathbf{\mathsf{ABC}}{-}\mathbf{\mathsf{XYZ}}$
`\bi`	`% sans serif bold slanted`	$\boldsymbol{\mathsf{abc}}{-}\boldsymbol{\mathsf{xyz}}, \boldsymbol{\mathsf{ABC}}{-}\boldsymbol{\mathsf{XYZ}}$

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

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

You can use `\mathbox{`⟨*text*⟩`}` inside math mode. It behaves as `{\hbox{`⟨*text*⟩`}}` (i.e. the ⟨*text*⟩ is printed in horizontal non-math mode) but the size of the ⟨*text*⟩ is adapted to the context of math size (text or script or scriptscript).

1.4 Typical elements of the document

1.4.1 Chapters and sections

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

```
\tit Document title ⟨end of line⟩
\chap Chapter title ⟨end of line⟩
\sec Section title ⟨end of line⟩
\secc Subsection title ⟨end of line⟩
```

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

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

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

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

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

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

1.4.2 Another numbered objects

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

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

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

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

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

```
\caption/t The dependency of the computer-dependency on the age.
\cskip
\noindent\hfil\table{rl}{
    age   & value \crl\noalign{\smallskip}
    0--1  & unmeasured \cr
    1--6  & observable \cr
    6--12 & significant \cr
   12--20 & extremal \cr
   20--40 & normal \cr
   40--60 & various \cr
   60--$\infty$ & moderate}
```

This example produces:

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

age	value
0–1	unmeasured
1–6	observable
6–12	significant
12–20	extremal
20–40	normal
40–60	various
60–∞	moderate

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

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

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

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

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

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

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

Three independent numbers are used in this example. One for Theorems and Corollaries second for Definitions and third for Examples. The user can write `\theorem Let $M$ 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[⟨label⟩]{⟨text⟩}` or `\pgref[⟨label⟩]{⟨text⟩}`. If the ⟨*text*⟩ includes the `@` character, it is replaced by implicitly printed text. Example: `see \ref[lab]{section~@}` prints the same as `see section~\ref[lab]`, but first case creates larger active area for mouse clicking, when `\hyperlinks` are declared.

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

You can use the `\label[⟨label⟩]` before the `\theorem`, `\definition` etc. (macros defined with `\numberedpar`) if you want to reference these numbered objects. You can't use `\theorem[⟨label⟩]` because the optional parameter is reserved to another purpose here.

You can create a reference to whatever else by commands `\label[⟨label⟩]\wlabel{⟨text⟩}`. The connection between ⟨*label*⟩ and ⟨*text*⟩ is established. The `\ref[⟨label⟩]` will print ⟨*text*⟩.

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

1.4.4 Hyperlinks, outlines

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

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

The last object is an external link and it is colored by ⟨*color-out*⟩. Other links are internal and they are colored by ⟨*color-in*⟩. Example:

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

You can use another marking of active links: by frames which are visible in the PDF viewer but invisible when the document is printed. The way to do it is to define the macros `\_pgborder`, `\_tocborder`, `\_citeborder`, `\_refborder` and `\_urlborder` as the triple of RGB components of the used color. Example:

```
\def\_tocborder {1 0 0}  % links in table of contents: red frame
\def\_pgborder {0 1 0}   % links to pages: green frame
\def\_citeborder {0 0 1} % links to references: blue frame
```

By default, these macros are not defined. It means that no frames are created.

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

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

The `\url` macro prints its parameter in `\tt` font and creates a potential breakpoints in it (after slash or dot, for example). If the `\hyperlinks` declaration is used then the parameter of `\url` is treated as an external URL link. An example: `\url{http://www.olsak.net}` creates `http://www.olsak.net`. The characters `%`, `\`, `#`, `{`, and `}` have to be protected by backslash in the `\url` argument, the other special characters `~`, `^`, `&` can be written as single character[4]. You can insert the `\|` command in the `\url` argument as a potential breakpoint.

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

The PDF format provides *outlines* which are notes placed in the special frame of the PDF viewer. These notes can be managed as a structured and hyperlinked table of contents of the document. The command `\outlines{`⟨*level*⟩`}` creates such outlines from data used for the table of contents in the document. The ⟨*level*⟩ parameter gives the level of opened sub-outlines in the default view. The deeper levels can be opened by mouse click on the triangle symbol after that.

If you are using a special unprotected macro in section titles then `\outlines` macro may crash. You must declare a variant of the macro for outlines case which is expandable. Use `\regmacro` in this case. See the section 1.5.1 for more information about `\regmacro`.

The command `\insertoutline{`⟨*text*⟩`}` inserts a next entry into PDF outlines at the main level 0. These entries can be placed before the table of contents (created by `\outlines`) or after it. Their hyperlink destination is in the place where the `\insertoutline` macro is used.

The command `\thisoutline{`⟨*text*⟩`}` uses ⟨*text*⟩ in the outline instead of default title text for the first following `\chap`, `\sec`, or `\secc`. Special case: `\thisoutline{\relax}` doesn't create any outline for the following `\chap`, `\sec`, or `\secc`.

1.4.5 Lists

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

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

---

[4] More exactly, there are the same rules as for `\code` command, see section 1.4.7.

```
\style A % items of type A), B), C), ...
\style x % small rectangle
\style X % big rectangle
\style d % definition list, use *{word}, see OpTeX trick 0108
```

For example:

```
\begitems
* First idea
* Second idea in subitems:
  \begitems \style i
  * First sub-idea
  * Second sub-idea
  * Last sub-idea
  \enditems
* Finito
\enditems
```

produces:

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

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

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

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

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

1.4.6 Tables

The macro `\table{`⟨*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||}{                    \crl
   Month    & commodity  & price    \crli \tskip2pt
   January  & notebook   & \$ 700   \cr
   February & skateboard & \$ 100   \cr
   July     & yacht      & k\$ 170  \crl}
```

generates the result:

Month	commodity	price
January	notebook	\$ 700
February	skateboard	\$ 100
July	yacht	k\$ 170

Apart from `l`, `r`, `c` declarators, you can use the `p{`⟨*size*⟩`}` declarator which declares the column with paragraphs of given width. More precisely, a long text in the table cell is printed as a multiline paragraph with given width. By default, the paragraph is left-right justified. But there are alternatives:

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

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

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

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

- `\crl` … the end of the row with a horizontal line after it.
- `\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{`⟨*list*⟩`}` … like `\crli` but the lines are drawn only in the columns mentioned in comma-separated ⟨*list*⟩ of their numbers. The ⟨*list*⟩ can include ⟨*from*⟩`-`⟨*to*⟩ declarators, for example `\crlp{1-3,5}` is equal to `\crlp{1,2,3,5}`.

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

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

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

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

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

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

```
\frame{\table{|c||l||r|}{ \crl
  \mspan3[|c|]{\bf Title} \crl   \noalign{\kern\hhkern}\crli
  first & second & third  \crlli
  seven & eight  & nine   \crli}}
```

creates the following result:

Title		
first	second	third
seven	eight	nine

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:

```
\thistable{\tabstrut={\vrule height 20pt depth10pt width0pt}
           \baselineskip=20pt \tablinespace=0pt \rulewidth=.8pt}
\table{|8{c|}}{\crlp{3-8}
   \mspan2[c|]{}      & \mspan3[c|]{Singular}         & \mspan3[c|]{Plural} \crlp{3-8}
   \mspan2[c|]{}      & Neuter & Masculine & Feminine & Masculine & Feminine & Neuter \crl
   \vspan2{I}   & Inclusive & \mspan3[c|]{\vspan2{O}} & \mspan3[c|]{X} \crlp{2,6-8}
                & Exclusive & \mspan3[c|]{}           & \mspan3[c|]{X} \crl
   \vspan2{II}  & Informal  & \mspan3[c|]{X}          & \mspan3[c|]{X} \crlp{2-8}
                & Formal    & \mspan6[c|]{X} \crl
   \vspan2{III} & Informal  & \vspan2{O} & X & X      & \mspan2[c|]{X} &\vspan2{O} \crlp{2,4-7}
                & Formal    &                         & \mspan4[c|]{X} & \crl
}
```

		Singular			Plural		
		Neuter	Masculine	Feminine	Masculine	Feminine	Neuter
I	Inclusive	O			X		
	Exclusive				X		
II	Informal	X			X		
	Formal	X					
III	Informal	O	X	Feminine X	X		O
	Formal		X				

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`s and `\hrule`s 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. OpTeX 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. The group ends by `\endtt`. You can declare settings valid only inside this group at the same line where `\begtt` starts. For example:

```
\begtt \adef!{?} \adef?{!}
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 defined character as the given macro.

You can declare settings valid inside each `\begtt`...`\endtt` group by the `\everytt` register. There are tips for such global `\everytt` declarations 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
```

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

The in-line verbatim surrounded by a `\verbchar` doesn't work in parameter of macros and macro definitions. (It works in titles declared by `\chap`, `\sec` etc. and in `\fnote`s, because these macros are specially defined in OpTeX). You can use more robust command `\code{`⟨*text*⟩`}` in problematic situations, but you have to escape the following characters in the ⟨*text*⟩: `\`, `#`, `%`, braces (if the braces are unmatched in the ⟨*text*⟩), and space or `^` (if there are more than one subsequent spaces or `^` in the ⟨*text*⟩). Examples:

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

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

```
\verbinput (12-42) program.c  % listing from program.c, only lines 12-42
\verbinput (-60) program.c    % print from begin to the line 60
\verbinput (61-) program.c    % from line 61 to the end
\verbinput (-) program.c      % whole file is printed
\verbinput (70+10) program.c  % from line 70, only 10 lines printed
\verbinput (+10) program.c    % from the last line read, print 10 lines
\verbinput (-5+7) program.c   % from the last line read, skip 5, print 7
\verbinput (+) program.c      % from the last line read to the end
```

You can insert additional commands for `\verbinput` before the first opening bracket. They are processed in the local group. For example, `\verbinput \hsize=20cm (-) program.c`.

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

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

The `\begtt...\endtt` pair or `\verbinput` can be used for listings of codes. Automatic syntax highlighting is possible, for example `\begtt \hisyntax{C}` activates colors for C programs. Or `\verbinput \hisyntax{HTML} (-) file.html` can be used for HTML or XML codes. OpTEX implements syntax highlighting of C, Lua, Python, TEX, HTML, XML and more. For a declaration of a new language, see the section 2.28.2.

If the code is read by `\verbinput` and there are comment lines prefixed by two characters then you can set them by `\commentchars`⟨*first*⟩⟨*second*⟩. Such comments are fully interpreted by TEX (i.e. not verbatim). Section 2.28.1 (page 145) says more about this feature.

1.5 Autogenerated lists

1.5.1 Table of contents

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

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

```
\nonum\notoc\sec Table of Contents
```

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

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

```
% default value of \mylogo macro used in text and in the titles:
\def\mylogo{\leavevmode\hbox{{\Red\it My}{\setfontsize{mag1.5}\rm Lo}Go}}
% another variants:
\regmacro {\def\mylogo{\hbox{\Red My\Black LoGo}}} % used in TOC
          {\def\mylogo{\hbox{{\it My}\/LoGo}}}     % used in running heads
          {\def\mylogo{MyLoGo}}                    % used in PDF outlines
```

1.5.2 Making the index

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

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

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

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

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

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

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

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

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

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

The number of the parts of one index entry (separated by `/`) is unlimited. Note, that you can spare your typing by the comma in the `\ii` parameter. The previous example is equivalent to `\ii linear/space \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. OpTeX 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. OpTeX provides other macros `\begmulti` and `\endmulti` for more columns:

`\begmulti` ⟨*number of columns*⟩
⟨*text*⟩
`\endmulti`

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

```
\sec Index
\begmulti 3 \makeindex \endmulti
```

Only “pure words” can be propagated to the index by the `\ii` command. It means that there cannot be any macro, TeX primitive, 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.
...
\iis chiquadrat {$\chi$-quadrat}
\iis relax {\code{\\relax}}
```

The `\iis` ⟨*equivalent*⟩ `{`⟨*text*⟩`}` creates one entry in the “dictionary of the exceptions”. The sorting is done by the ⟨*equivalent*⟩ but the ⟨*text*⟩ is printed in the index entry list.

The sorting rules when `\makeindex` runs depends on the current language. See section 1.7.1 about languages selection. The < character is sorted before A and > after Z and these characters aren’t printed if they are first in the index item.

1.5.3 BibTEXing

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

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

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

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

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

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

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

- Manually using `\bib[`⟨*label*⟩`]` commands.
- By `\usebib/`⟨*type*⟩ `(`⟨*style*⟩`)` ⟨*bib-base*⟩ command which reads `*.bib` files directly.

Note that the other two possibilities documented in OPmac (using external BibTEX program) aren't supported because BibTEX is an old program that does not support Unicode. And Biber seems to be not compliant with Plain TEX.

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

```
\bib [tbn] P. Olšák. {\it\TeX{}book naruby.} 468~s. Brno: Konvoj, 1997.
\bib [tst] P. Olšák. {\it Typografický systém \TeX.}
           269~s. Praha: CSTUG, 1995.
```

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

```
\bib [tbn] = {Olšák, 2001}
    P. Olšák. {\it\TeX{}book naruby.} 468~s. Brno: Konvoj, 2001.
```

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

```
\usebib/c (⟨style⟩) ⟨bib-base⟩ % sorted by \cite-order (c=cite),
\usebib/s (⟨style⟩) ⟨bib-base⟩ % sorted by style (s=style).
% example:
\nocite[*] \usebib/s (simple) op-biblist  % prints all from op-biblist.bib
```

The ⟨*bib-base*⟩ is one or more `*.bib` database source files (separated by commas and without extension) and the ⟨*style*⟩ is the part of the filename `bib-`⟨*style*⟩`.opm` where the formatting of the references list is defined. OpTEX supports `simple` or `iso690` styles. The features of the `iso690` style is documented in the section 2.32.6 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`s and references list are printed (and interconnected) independent of another parts of your document. The `\cite` labels used in different parts can be the same and they are not affected. References lists can be created manually by `\bib` or from a database by `\usebib`. Example:

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

By default, `\bibpart` is empty. So `\cite`s 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 LaTeX file `x11nam.def`. There are about 300 color names like `\DeepPink`, `\Chocolate` etc. If there are numbered variants of the same name, then the letters B, C, etc. are appended to the name in OpTEX. For example `\Chocolate` is Chocolate1, `\ChocolateB` is Chocolate2 etc.

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

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

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

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

---

[5] Printed output is more equal to the monitor preview especially if you are using ICC profile for your printer.

it is truncated to 1. If a convex linear combination (as in the last example above) is used then it emulates color behavior on a painter's palette. You can use `\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 ⟨*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*⟩`.`⟨*extension*⟩ to the document. You can set the picture width by `\picw=`⟨*dimen*⟩ before `\inspic` command which declares the width of the picture. The image files can be in the PNG, JPG, JBIG2 or PDF format.

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

The image files are searched in `\picdir`. This token list is empty by default, this means that the image files are searched in the current directory. Example: `\picdir={img/}` supposes that image files are in `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 TeXto print labels in the same font as document font. OpTeX supports this feature by `\inkinspic` `{`⟨*filename*⟩`.pdf}` command. It reads and displays both: PDF image and labels generated by Inkscape.

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

1.6.3 PDF transformations

All typesetting elements are transformed by linear transformation given by the current transformation matrix. The `\pdfsetmatrix` `{`⟨*a*⟩ ⟨*b*⟩ ⟨*c*⟩ ⟨*d*⟩`}` command makes the internal multiplication with the current matrix so linear transformations can be composed. One linear transformation given by the `\pdfsetmatrix` above transforms the vector $[0,1]$ to $[\langle a\rangle,\langle b\rangle]$ 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.

---

[6] A powerful and free Wysiwyg editor for creating vector graphics.

[7] Chose "Omit text in PDF and create LaTeX file" option.

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

```
\pdfscale{⟨horizontal-factor⟩}{⟨vertical-factor⟩}
\pdfrotate{⟨angle-in-degrees⟩}
```

These macros simply call the properly `\pdfsetmatrix` command.

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

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

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

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

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

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

1.6.4 Ovals, circles

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

```
\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-righ margin, top-bottom margin
```

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

If you need to set a parameters for the ⟨*text*⟩ (color, size, font etc.), put such setting right in front of the ⟨*text*⟩: `\inoval{⟨text settings⟩⟨text⟩}`.

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

```
\circleparams={\ratio=1 \fcolor=\Yellow \lcolor=\Red \lwidth=0.5bp
               \shadow=N \overlapmargins=N \hhkern=2pt \vvkern=2pt}
```

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

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

1.6.5 Putting images and texts wherever

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

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

1.7 Others

1.7.1 Using more languages

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

en(USEnglish) enus(USenglishmax) engb(UKenglish) be(Belarusian) bg(Bulgarian) ca(Catalan) hr(Croatian) cs(Czech) da(Danish) nl(Dutch) et(Estonian) fi(Finnish) fis(schoolFinnish) fr(French) de(nGerman) deo(oldGerman) gsw(swissGerman) elm(monoGreek) elp(Greek) grc(ancientGreek) hu(Hungarian) is(Icelandic) ga(Irish) it(Italian) la(Latin) lac(classicLatin) lal(liturgicalLatin) lv(Latvian) lt(Lithuanian) mk(Macedonian) pl(Polish) pt(Portuguese) ro(Romanian) rm(Romansh) ru(Russian) srl(Serbian) src(SerbianCyrl) sk(Slovak) sl(Slovenian) es(Spanish) sv(Swedish) uk(Ukrainian) cy(Welsh) af(Afrikaans) hy(Armenian) as(Assamese) eu(Basque) bn(Bengali) nb(Bokmal) cop(Coptic) cu(churchslavonic) eo(Esperanto) ethi(Ethiopic) fur(Friulan) gl(Galician) ka(Georgian) gu(Gujarati) hi(Hindi) id(Indonesian) ia(Interlingua) kn(Kannada) kmr(Kurmanji) ml(Malayalam) mr(Marathi) mn(Mongolian) nn(Nynorsk) oc(Occitan) or(Oriya) pi(Pali) pa(Panjabi) pms(Piedmontese) zh(Pinyin) sa(Sanskrit) ta(Tamil) te(Telugu) th(Thai) tr(Turkish) tk(Turkmen) hsb(Uppersorbian) he(Hebrew)

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

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

You can switch between language patterns by \⟨*iso-code*⟩`lang` commands mentioned above. Default is `\enlang`.

OpTeX generates three phrases used for captions and titles in technical articles or books: "Chapter", "Table" and "Figure". These phrases need to be known in used language and it depends on the previously used language selectors \⟨*iso-code*⟩`lang`. OpTeX declares these words only for few languages: Czech, German, Spanish, French, Greek, Italian, Polish, Russian, Slovak, Hebrew and English, If you need to use these words in other languages or you want to auto-generate more words in your macros, then you can declare it by `\sdef` or `\_langw` commands as shown in section 2.37.2.

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

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

1.7.2 Pre-defined styles

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

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

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

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

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

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

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

1.7.3 Loading other macro packages

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

```
\load [qrcode, scanbase]
```

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

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

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

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

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

1.7.4 Lorem ipsum dolor sit

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

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

1.7.5 Logos

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

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

1.7.6 The last page

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

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

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

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

1.7.7 Use OpTEX

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

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

1.7.8 OpTEX tricks

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

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

`\algol`, `\algolkw`, `\algolcap`, `\makeLOA`, `\algolnum` enables to create pseudocode listings.
`\beglua`, `\begLUA`, `\logginglua` writing LUA codes as LUA codes.
`\cancel` prints a given text and the line/cross line over the text.
`\clipbox` creates a box of given dimensions with clipped content.
`\colortab` colored cells in the table.
`\correctvsize` sets `\vsize` to fit lines exactly to pages.
`\createfile`, `\begfile`, `\endfile` writes a code from the document to the given file.
`\crtop`, `\crmid`, `\crbot` specific design of tables: only horizontal rules with different thickness.
`\crx` alternating colored lines in tables.
`\csvread` reads CSV databases and provides other macros for working with their data.
`\dbitem`, `\dbwhere`. `\dbloop`, `\dbsort`, `\dbcreate`, `\dbinsert`, `\dbshow`. etc., works with databases.
`\directchar` prints the character directly, bypasses the ligature processing.
`\directoutput` puts boxes to standalone pages adatped to the box dimensions.
`\easylist` the depth of list is given by the number of `*`.
`\fcread`, `\fullcite` citations by full bibliographic records.
`\framedblocks` redefines `\begblock`, `\endblock` to create blocks in frames splittable to pages.
`\ignoreinspic` the `\inspic` commands stop loading images, they are replaced by gray frames.
`\import` allows to have subsets of document input files in separate directories.
`\ismatch`, `\isinmacro` tests, if the given string or the macro body includes the given substring.
`\ispageodd` tests, if the current point is at odd page regardless of asynchronous processing.
`\incrpp`, `\thepp`, `\thepplast`, `\truepage` does per-page counting of objects.
`\keystroke` prints given text in a keystroke-like frame.
`\longtable` allows to break a table to more pages and repeats header.
`\makeLOF`, `\makeLOT`, `\captionF`, `\captionT` create list of tables and list of figures similar to `\maketoc`.
`\mathinexpr` allows short names `sin(x)`, `log(x)` instead `math.sin(x)`, `math.log(x)` in `\expr` arguments.
`\onlyifnew` only define a macro if it is not already defined.
`\pgforeground` adds material to the foreground of each page.
`\pstart`, `\pend` displays line numbers of the marked text in the margin.
`\rebox` modifies the vbox: its width will be equal to the wider line.
`\replmacro` enables to patch existing macros using regular expression rules.
`\roundframe` colored frames with rounded corners and many options.
`\runsystem` runs the given external system command.
`\shadedframe` colored rectangular frames with simple shadows.
`\scaleto`, `\scaletof` text font size changed to the desired width.
`\seccc`, `\iniseccc` implements new level of subsubsections.
`\shownodes` prints the list of nodes to the terminal.
`\sethours`, `\setminutes`, `\setseconds`, `\setweekday` printing time, date, and day of week.
\style m, `\keepstyle` creates lists with items numbered like subsections.
`\settabs`, `\tabs` macros emulate tabulators of old typewriters.
`\showpglists` shows good organized list of nodes of given pages to the log file.
`\tabnodes` positions of table items are nodes, they can be used for drawing.
`\tdnum` expands to the three-digits-group format of the given number.
`\tnote` creates notes for table data printed just after the table.
`\treedata`, `\tracingtreedata`, `\treenodes` `\showtree`, `\jsonread`, support tree data structures.
`\ttlineref` verbatim lines referenced in text.
`\vcent`, `\vbot` prints paragraphs in tables vertically centered or placed at bottom.
`\thedimen` prints dimen value using selected unit.
`\twoblocks` allows printing bilingual texts in two columns veritically aligned.
`\xparshape` behaves like `\parshape` but its declaration is valid for more paragraphs.
`\xreplstring`, `\ereplstring` behave like `\replstring` but are expandable and add more features.

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

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

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

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

\hyperlinks {color-in}{color-out} % PDF links activate as clickable
\outlines {level}   % PDF will have a table of contents in the left tab
\magscale[factor]  % resize typesetting, line/page breaking unchanged
\margins/pg format (left, right, top, bottom)unit % margins setting
\report \letter \slides  % style declaration macros
```

1.9 API for macro writers

All TeX primitives and almost all OpTeX macros are accessible by two names: `\foo` (public or user namespace) and `\_foo` (private name space). For example `\hbox` and `\_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 `\addto` macro below. The `\_addto` 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 `\_codedecl` macro below, so this macro is not available as `\codedecl`.

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 `\_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, `\aheadto \macro{`⟨*text*⟩`}` puts ⟨*text*⟩ at the begin.
`\adef` ⟨*char*⟩`{`⟨*body*⟩`}` defines ⟨*char*⟩ active character with meaning ⟨*body*⟩.
`\afterfi {`⟨*text*⟩`}`⟨*ignored*⟩`\fi` expands to `\fi`⟨*text*⟩, see also `\afterxfi`.
`\basefilename \currfile` returns the name of the file currently read.
`\bp {`⟨*dimen expression*⟩`}` expands TeX dimension to decimal number in `bp` without unit.
`\casesof` ⟨*token*⟩ ⟨*list of cases*⟩ expands to a given case by the given ⟨*token*⟩. See also `\qcasesof`, `\xcasesof`.
`\_codedecl` ⟨*sequence*⟩ `{`⟨*info*⟩`}` is used at beginning of macro files.
`\colordef \macro {`⟨*mix of colors*⟩`}` declares `\macro` as color switch.
`\cs {`⟨*string*⟩`}` expands `\`⟨*string*⟩.
`\cstochar` ⟨*sequence*⟩ converts ⟨*sequence*⟩ to ⟨*character*⟩ if there was `\let`⟨*sequence*⟩`=`⟨*character*⟩.
`\_doc ... \_cod` encloses documentation text in the macro code.
`\eoldef \macro #1{`⟨*body*⟩`}` defines `\macro` with parameter separated to end of line.
`\_endcode` closes the part of macro code in macro files.
`\_endnamespace` closes name space declared by `\_namespace`.
`\eqbox [`⟨*label*⟩`]{`⟨*text*⟩`}` creates `\hbox{`⟨*text*⟩`}` with common width across whole document.
`\expr {`⟨*expression*⟩`}` expands to result of the ⟨*expression*⟩ with decimal numbers.
`\fontdef \f {`⟨*font spec.*⟩`}` declares `\f` as font switch.
`\fontlet \fa=\fb` ⟨*sizespec.*⟩ declares `\fa` as the same font switch like `\fb` at given ⟨*sizespec.*⟩.
`\foreach` ⟨*list*⟩`\do` ⟨*parameters*⟩`{`⟨*what*⟩`}` is exapandable loop over ⟨*list*⟩.
`\foreachdef \macro` ⟨*parameters*⟩`{`⟨*what*⟩`}` declares expandable `\macro` as loop over ⟨*list*⟩.
`\fornum` ⟨*from*⟩`..`⟨*to*⟩`\do {`⟨*what*⟩`}` is expanadable loop with numeric variable.
`\incr` ⟨*counter*⟩ increases and `\decr` ⟨*counter*⟩ decreases ⟨*counter*⟩ by one globally.
`\ignoreit` ⟨*one*⟩, `\ignoresecond` ⟨*one*⟩⟨*two*⟩ ignores given parameter.
`\expandafter \ignorept \the`⟨*dimen*⟩ expands to decimal number ⟨*dimen*⟩ without `pt`.
`\isempty`, `\istoksempty`, `\isequal`, `\ismacro`, `\isdefined`, `\isinlist \isfile`, `\isfont` do various tests.
Example: `\isinlist\list{`⟨*text*⟩`}\iftrue` does `\iftrue` if ⟨*text*⟩ is in `\list`.
`\isnextchar` ⟨*char*⟩`{`⟨*text1*⟩`}{`⟨*text2*⟩`}` performs ⟨*text1*⟩ if next character is ⟨*char*⟩, else ⟨*text2*⟩.
`\kv {`⟨*key*⟩`}` expands to a value given by key=value. See also `\trykv`, `\iskv`, `\readkv`, `\kvx`, `\nokvx`.
`\loop ... \repeat` is classical Plain TeX loop. `\xloop ...\do ... \repeat` is an extension of `\loop`.
`\mathstyles {`⟨*math list*⟩`}` enables to create macros dependent on current math style.
`\_namespace {`⟨*pkg*⟩`}` declares name space used by package writers.
`\newcount`, `\newdimen` etc. are classical Plain TeX allocators.
`\newif \iffoo` declares boolean `\iffoo` as in Plain TeX.
`\_newifi \_iffoo` declares boolean `\_iffoo`.
`\nospaceafter\macro`, `\nospacefuturelet`: they ignore the following optional space.
`\opinput {`⟨*filename*⟩`}` reads file like `\input` but with standard catcodes.
`\optdef \macro [`⟨*opt-default*⟩`]` ⟨*parameters*⟩`{`⟨*body*⟩`}` defines `\macro` with [opt.parameter].
`\opwarning {`⟨*text*⟩`}` prints ⟨*text*⟩ to the terminal and .log file as warning.
`\posx[`⟨*label*⟩`]`, `\posy[`⟨*label*⟩`]`, `\pospg[`⟨*label*⟩`]` provide coordinates of absolute position of the `\setpos[`⟨*label*⟩`]`.
`\private` ⟨*sequence*⟩ ⟨*sequence*⟩ `... ;` declares ⟨*sequence*⟩s for private name space.
`\public` ⟨*sequence*⟩ ⟨*sequence*⟩ `... ;` declares ⟨*sequence*⟩s for public name space.
`\replstring \macro{`⟨*stringA*⟩`}{`⟨*stringB*⟩`}` replaces all ⟨*stringA*⟩ to ⟨*stringB*⟩ in `\macro`.
`\sdef {`⟨*string*⟩`}`⟨*parameters*⟩`{`⟨*body*⟩`}` behaves like `\def\`⟨*string*⟩⟨*parameters*⟩`{`⟨*body*⟩`}`.
`\setctable` and `\restorectable` manipulate with stack of catcode tables.
`\slet {`⟨*stringA*⟩`}{`⟨*stringB*⟩`}` behaves like `\let\`⟨*stringA*⟩`=\`⟨*stringB*⟩
`\sxdef {`⟨*string*⟩`}`⟨*parameters*⟩`{`⟨*body*⟩`}` behaves like `\xdef\`⟨*string*⟩⟨*parameters*⟩`{`⟨*body*⟩`}`.
`\trycs {`⟨*string*⟩`}{`⟨*text*⟩`}` expands `\`⟨*string*⟩ if it is defined else expands ⟨*text*⟩.
`\useit` ⟨*one*⟩, `\usesecond` ⟨*one*⟩⟨*two*⟩ uses given parameter.
`\wlog {`⟨*text*⟩`}` writes ⟨*text*⟩ to .log file.

`\wterm` `{`⟨*text*⟩`}` writes ⟨*text*⟩ to the terminal and .log file.
`\xargs` ⟨*what*⟩ ⟨*token*⟩ ⟨*token*⟩ `...` `;` repeats ⟨*what*⟩⟨*token*⟩ for each ⟨*token*⟩.

1.10 Compatibility with Plain 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 `\protected`.

The text accents macros `\"`, `\'`, `\v`, `\u`, `\=`, `\^`, `\.`, `\H`, `\~`, `` \` ``, `\t` are undefined[8] in OpTeX. Use real letters like á, ř, ž in your source document instead of these old accents macros. If you really want to use them, you can initialize them by the `\oldaccents` command. But we don't recommend it.

The default paper size is not set as the letter with 1 in margins but as A4 with 2.5 cm 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 1 in 1 in coordinates but at the top left paper corner exactly. For example, `\hoffset` includes directly left margin.

The tabbing macros `\settabs` and `\+` (from Plain TeX) are not defined in 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[9].

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.

---

[8] The math accents macros like `\acute`, `\bar`, `\dot`, `\hat` still work.

[9] Use `$\secant(x)$` to get $\sec(x)$.

Chapter 2

Technical documentation

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

```
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.

`optex.ini`

```
%% This is part of the OpTeX project, see http://petr.olsak.net/optex

%% OpTeX ini file
%% Petr Olsak <project started from: Jan. 2020>
```

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.

`optex.ini`

```
% Catcodes:

\catcode `\{=1 % left brace is begin-group character
\catcode `\}=2 % right brace is end-group character
\catcode `\$=3 % dollar sign is math shift
\catcode `\&=4 % ampersand is alignment tab
\catcode `\#=6 % hash mark is macro parameter character
\catcode `\^=7 %
\catcode `\^^K=7 % circumflex and uparrow are for superscripts
\catcode `\^^A=8 % downarrow is for subscripts
\catcode `\^^I=10 % ascii tab is a blank space
\catcode `\_=11 % underline can be used in control sequences
\catcode `\~=13 % tilde is active
\catcode `\^^a0=13 % non breaking space in Unicode
\catcode 127=12 % normal character
```

The `\optexversion` and `\fmtname` are defined.

`optex.ini`

```
% OpTeX version

\def\optexversion{1.18 May 2025}
\def\fmtname{OpTeX}
\let\fmtversion=\optexversion
```

We check if LuaTEX engine is used at `-ini` state. And the `^^J` character is set as `\newlinechar`.

optex.ini

```
% Engine testing:

\newlinechar=`\^^J
\ifx\directlua\undefined
   \message{This format is based only on LuaTeX, use luatex -ini optex.ini^^J}
   \endinput \fi

\ifx\bgroup\undefined \else
   \message{This file can be used only for format initialisation, use luatex -ini^^J}
   \endinput \fi
```

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

optex.ini

```
% Basic .opm syntax:

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

Individual `*.opm` macro files are read.

optex.ini

```
% Initialization:

\message{OpTeX (Olsak's Plain TeX) initialization <\optexversion>^^J}

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

The file `optex.lua` is embedded into the format as byte-code. It is documented in section 2.39.

optex.ini

```
\_directlua{
   % preload OpTeX's Lua code into format as bytecode
   lua.bytecode[1] = assert(loadfile(kpse.find_file("optex", "lua")))
}
```

The `\everyjob` register is initialized and the format is saved by the `\dump` command.

optex.ini

```
\_everyjob = {%
   \_message{\_banner^^J}%
   \_directlua{lua.bytecode[1]()}% load OpTeX's Lua code
   \_mathsbon % replaces \int_a^b to \int _a^b
   \_inputref % inputs \jobname.ref if exists
}

\dump % You can redefine \dump if additional macros are needed. Example:
      % \let\dump=\relax \input optex.ini \input mymacros \_dump
```

2.2 Basic principles of OpTeX sources

2.2.1 Concept of namespaces of control sequences

OpTeX sets the category code of the "`_`" character to 11 (letter) and it is never changed.[1] So, we can always construct multiletter control sequence names from letters `A`–`Z`, `a`–`z`, and `_`. The "letter `_`" works in math mode as a subscript constructor because it is set as math active character (see section 2.15).

We distinguish following namespaces for multiletter control sequences:

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

All TeX primitives are initialized with two control sequences with the same meaning: *prefixed* control sequence (in private namespace, for example `\_hbox`) and *unprefixed* control sequence (in public namespace, for example `\hbox`). All OpTeX macros intended for end users are initialized in these two forms too, for example `\_ref` and `\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 `\def\fi{finito}` and nothing bad happens, if the user doesn't use `\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 name space, 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 (`\_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 `\public` macro:

`\public` ⟨*list of control sequences*⟩ `;`

For example `\public \foo \bar ;` does `\let\foo=\_foo`, `\let\bar=\_bar`.

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

---

[1] This is only singular exception form 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 unprefixed variant of a control sequence is declared already but we need the prefixed variant too.

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

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    ... is interpreted as \int _a^b
\max_M      ... is interpreted as \max _M
\alpha_{ij} ... 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 `\mathsboff`. After this, you can still write `$∫_a^b$` (Unicode) or `$\int _a^b$` without problems but `\int_a^b` yields to undefined control sequence `\int_a`. You can activate this feature again by `\mathsbon`. The effect will take shape from next line read from input file.

2.2.2 Macro files syntax

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

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

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

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

```
@:[⟨file name⟩] ⟨short title⟩ <⟨version⟩>
```

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

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

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

```
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 `\_`⟨*pgk*⟩`_`⟨*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.[2] 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

---

[2] We have not adopted the idea from expl3 language:)

write (and backward can read his/her code) simply with `\.foo`, `\.bar` control sequences and `\_⟨pkg⟩_foo`, `\_⟨pkg⟩_bar` control sequences are processed internally. The scope of the `\_namespace` command ends at the `\_endnamespace` command or when another `\_namespace` is used. This command checks if the same package label is not declared by the `\_namespace` twice.

`\_nspublic` ⟨*list of sequences*⟩ `;` does `\let\foo = \_⟨pkg⟩_foo` for each given sequence when `\_namespace{⟨pkg⟩}` is declared. Moreover, it prints a warning if `\foo` is defined already. The `\_nsprivate` macro does reverse operation to it without warnings. Example: you can define `\def\.macro{...}` and then set it to the public namespace by `\_nspublic \macro;`.

It could happen that a package writer needs to declare a control sequence (say `\foo`) directly without setting it in `\_⟨pkg⟩_foo` namespace followed by using `\_nspublic`. The `\newpublic` prefix should be used in this case, for example `\_newpublic\_def\foo` or `\_newpublic\_chardef\foo` 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 {⟨pkg⟩}` is initialized. Or close your namespace by `\_endnamespace` and open it again (after other packages are loaded) by `\_resetnamespace {⟨pkg⟩}`.

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

2.2.4 Summary about rules for external macro files published for OpTeX

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

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

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

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

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

prefixed.opm

```
\_codedecl \public {Prefixing and code syntax <2024-02-02>} % preloaded in format
```

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

prefixed.opm

```
\let\_directlua = \directlua
\_directlua {
    % enable all TeX primitives with _ prefix
    tex.enableprimitives('_', tex.extraprimitives('tex'))
    % enable all primitives without prefixing
    tex.enableprimitives('', tex.extraprimitives())
    % enable all primitives with _ prefix
    tex.enableprimitives('_', tex.extraprimitives())
}
```

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

`\public` `\`⟨*sequence*⟩ `\`⟨*sequence*⟩ `... ;` does `\let` `\`⟨*sequence*⟩ `=` `\_`⟨*sequence*⟩ for all sequences.

`\private` `\`⟨*sequence*⟩ `\`⟨*sequence*⟩ `...;` does `\let` `\_`⟨*sequence*⟩ `=` `\`⟨*sequence*⟩ for all sequences.

`\newpublic`⟨*do*⟩`\`⟨*sequence*⟩ prints warning if `\`⟨*sequence*⟩ is declared already. Then runs ⟨*do*⟩`\`⟨*sequence*⟩.

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

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

prefixed.opm

```
\_let\_ea =\_expandafter  % useful shortcut

\_long\_def \_xargs #1#2{\_ifx #2;\_else \_ea#1\_ea#2\_ea\_xargs \_ea #1\_fi}

\_def \_pkglabel{}
\_def \_public {\_xargs \_publicA}
\_def \_publicA #1{%
   \_checkexists \public {_\_csstring#1}%
   \_ea\_let \_ea#1\_csname  _\_csstring #1\_endcsname
}
\_def \_private {\_xargs \_privateA}
\_def \_privateA #1{%
   \_checkexists \private {\_csstring #1}%
   \_ea\_let \_csname  _\_csstring #1\_endcsname =#1%
}
\_def\_checkexists #1#2{\_unless \_ifcsname #2\_endcsname
   \_errmessage {\_string#1: \_bslash#2 must be declared}\_fi
}
\_def\_newpublic #1#2{\_unless\_ifx #2\_undefined
   \_opwarning{\_string#2 is redefined%
      \_ifx\_pkglabel\_empty \_else\_space by the \_ea\_ignoreit\_pkglabel\_space package\_fi}\_fi
   #1#2%
}
\_public \public \private \newpublic \xargs \ea ;
```

We define the macros `\_namespace` `{`⟨*pkg label*⟩`}`, `\_resetnamespace` `{`⟨*pkg label*⟩`}`, `\_endnamespace`, `\_pkglabel`, `\_nspublic`, and `\_nsprivate` for package writers, see section 2.2.3.

prefixed.opm

```
\_def \_pkglabel{}
\_def\_namespace #1{%
   \_ifcsname _namesp:#1\_endcsname \_errmessage
      {The name space "#1" is used already, it cannot be used twice}%
      \_endinput
   \_else
      \_ea \_gdef \_csname _namesp:#1\_endcsname {}%
      \_resetnamespace{#1}\_fi
}
\_def\_resetnamespace #1{%
   \_unless \_ifx \_pkglabel\_empty \_endnamespace \_fi
   \_gdef \_pkglabel{_#1}%
   \_directlua{
      callback.add_to_callback("process_input_buffer",
        function (str)
           return string.gsub(str, "\_nbb[.]([a-zA-Z])", "\_nbb _#1_\_pcent 1")
        end, "_namespace")
   }%
}
\_def\_endnamespace {%
```

```
   \_directlua{ callback.remove_from_callback("process_input_buffer", "_namespace") }%
   \_gdef \_pkglabel{}%
}
\_def \_nspublic {\_xargs \_nspublicA}
\_def \_nspublicA #1{%
   \_checkexists \_nspublic {\_pkglabel _\_csstring #1}%
   \_ifcsname _eol:\_ea\_ignoreit\_pkglabel _\_csstring #1\_endcsname % defined by \eoldef
      \_slet {_eol:\_csstring #1}{_eol:\_ea\_ignoreit\_pkglabel _\_csstring #1}\_fi
   \_ea\_newpublic \_ea\_let \_ea#1\_csname \_pkglabel _\_csstring #1\_endcsname
}
\_def \_nsprivate {\_xargs \_nsprivateA}
\_def \_nsprivateA #1{%
   \_checkexists \_nsprivate {\_csstring #1}%
   \_ea\_let \_csname \_pkglabel _\_csstring #1\_endcsname =#1%
}
```

Each macro file should begin with `\_codedecl` `\macro` `{`⟨*info*⟩`}`. If the `\macro` is defined already then the `\endpinput` protects to read such file more than once. Else the ⟨*info*⟩ is printed to the terminal and the file is read. The `\_endcode` is defined as `\endinput` in the `optex.ini` file. `\wterm` `{`⟨*text*⟩`}` prints the ⟨*text*⟩ to the terminal and to the `.log` file, `\wlog` `{`⟨*text*⟩`}` prints the ⟨*text*⟩ only to the `.log` file (as in plain TeX)

prefixed.opm

```
\_def \_codedecl #1#2{%
   \_ifx #1\_undefined \_wlog{@:[\_basefilename\_currfile] #2}%
   \_else \_ea \_endinput \_fi
}
\_def \_wterm {\_immediate \_write16 }
\_def \_wlog {\_immediate\_write-1 } % write on log file (only)

\_public \wterm \wlog ;
```

`\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.

prefixed.opm

```
\_def\_currfile{\_directlua{tex.print(status.filename)}}
\_def\_basefilename #1{\_ea\_nofileext\_expanded{\_ea\_ea\_ea\_nofilepath#1/\_fin}.\_fin}
\_def\_nofilepath #1/#2{\ifx#2\_fin #1\_else \_ea\_nofilepath \_ea#2\_fi}
\_def\_nofileext #1.#2\_fin{#1}

\_public \currfile \basefilename ;
```

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.

prefixed.opm

```
\_protected\_long \_def \_fin \_fin {}
```

2.3 pdfTeX initialization

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

luatex-ini.opm

```
\_codedecl \pdfprimitive {LuaTeX initialization code <2024-02-29>} % preloaded in format

\_let\_pdfpagewidth        \pagewidth
\_let\_pdfpageheight       \pageheight
\_let\_pdfadjustspacing    \adjustspacing
\_let\_pdfprotrudechars    \protrudechars
\_let\_pdfnoligatures      \ignoreligaturesinfont
\_let\_pdffontexpand       \expandglyphsinfont
\_let\_pdfcopyfont         \copyfont
\_let\_pdfxform            \saveboxresource
\_let\_pdflastxform        \lastsavedboxresourceindex
\_let\_pdfrefxform         \useboxresource
\_let\_pdfximage           \saveimageresource
\_let\_pdflastximage       \lastsavedimageresourceindex
```

```
\_let\_pdflastximagepages  \lastsavedimageresourcepages
\_let\_pdfrefximage        \useimageresource
\_let\_pdfsavepos          \savepos
\_let\_pdflastxpos         \lastxpos
\_let\_pdflastypos         \lastypos
\_let\_pdfoutput           \outputmode
\_let\_pdfdraftmode        \draftmode
\_let\_pdfpxdimen          \pxdimen
\_let\_pdfinsertht         \insertht
\_let\_pdfnormaldeviate    \normaldeviate
\_let\_pdfuniformdeviate   \uniformdeviate
\_let\_pdfsetrandomseed    \setrandomseed
\_let\_pdfrandomseed       \randomseed
\_let\_pdfprimitive        \primitive
\_let\_ifpdfprimitive      \ifprimitive
\_let\_ifpdfabsnum         \ifabsnum
\_let\_ifpdfabsdim         \ifabsdim

\_public
   \pdfpagewidth \pdfpageheight \pdfadjustspacing \pdfprotrudechars
   \pdfnoligatures \pdffontexpand \pdfcopyfont \pdfxform \pdflastxform
   \pdfrefxform \pdfximage \pdflastximage \pdflastximagepages \pdfrefximage
   \pdfsavepos \pdflastxpos \pdflastypos \pdfoutput \pdfdraftmode \pdfpxdimen
   \pdfinsertht \pdfnormaldeviate \pdfuniformdeviate \pdfsetrandomseed
   \pdfrandomseed \pdfprimitive \ifpdfprimitive \ifpdfabsnum \ifpdfabsdim ;

\_directlua {tex.enableprimitives('pdf',{'tracingfonts'})}

\_protected\_def \_pdftexversion     {\_numexpr 140\_relax}
           \_def \_pdftexrevision    {7}
\_protected\_def \_pdflastlink       {\_numexpr\_pdffeedback lastlink\_relax}
\_protected\_def \_pdfretval         {\_numexpr\_pdffeedback retval\_relax}
\_protected\_def \_pdflastobj        {\_numexpr\_pdffeedback lastobj\_relax}
\_protected\_def \_pdflastannot      {\_numexpr\_pdffeedback lastannot\_relax}
           \_def \_pdfxformname      {\_pdffeedback xformname}
           \_def \_pdfcreationdate   {\_pdffeedback creationdate}
           \_def \_pdffontname       {\_pdffeedback fontname}
           \_def \_pdffontobjnum     {\_pdffeedback fontobjnum}
           \_def \_pdffontsize       {\_pdffeedback fontsize}
           \_def \_pdfpageref        {\_pdffeedback pageref}
           \_def \_pdfcolorstackinit {\_pdffeedback colorstackinit}
\_protected\_def \_pdfliteral        {\_pdfextension literal}
\_protected\_def \_pdfcolorstack     {\_pdfextension colorstack}
\_protected\_def \_pdfsetmatrix      {\_pdfextension setmatrix}
\_protected\_def \_pdfsave           {\_pdfextension save\_relax}
\_protected\_def \_pdfrestore        {\_pdfextension restore\_relax}
\_protected\_def \_pdfobj            {\_pdfextension obj }
\_protected\_def \_pdfrefobj         {\_pdfextension refobj }
\_protected\_def \_pdfannot          {\_pdfextension annot }
\_protected\_def \_pdfstartlink      {\_pdfextension startlink }
\_protected\_def \_pdfendlink        {\_pdfextension endlink\_relax}
\_protected\_def \_pdfoutline        {\_pdfextension outline }
\_protected\_def \_pdfdest           {\_pdfextension dest }
\_protected\_def \_pdfthread         {\_pdfextension thread }
\_protected\_def \_pdfstartthread    {\_pdfextension startthread }
\_protected\_def \_pdfendthread      {\_pdfextension endthread\_relax}
\_protected\_def \_pdfinfo           {\_pdfextension info }
\_protected\_def \_pdfcatalog        {\_pdfextension catalog }
\_protected\_def \_pdfnames          {\_pdfextension names }
\_protected\_def \_pdfincludechars   {\_pdfextension includechars }
\_protected\_def \_pdffontattr       {\_pdfextension fontattr }
\_protected\_def \_pdfmapfile        {\_pdfextension mapfile }
\_protected\_def \_pdfmapline        {\_pdfextension mapline }
\_protected\_def \_pdftrailer        {\_pdfextension trailer }
\_protected\_def \_pdfglyphtounicode {\_pdfextension glyphtounicode }
\_protected\_def \_pdfrunninglinkoff {\_pdfextension linkstate 1 }
\_protected\_def \_pdfrunninglinkon  {\_pdfextension linkstate 0 }

\_protected\_edef\_pdfcompresslevel       {\_pdfvariable compresslevel}
```

```
\_protected\_edef\_pdfobjcompresslevel    {\_pdfvariable objcompresslevel}
\_protected\_edef\_pdfdecimaldigits       {\_pdfvariable decimaldigits}
\_protected\_edef\_pdfgamma               {\_pdfvariable gamma}
\_protected\_edef\_pdfimageresolution     {\_pdfvariable imageresolution}
\_protected\_edef\_pdfimageapplygamma     {\_pdfvariable imageapplygamma}
\_protected\_edef\_pdfimagegamma          {\_pdfvariable imagegamma}
\_protected\_edef\_pdfimagehicolor        {\_pdfvariable imagehicolor}
\_protected\_edef\_pdfimageaddfilename    {\_pdfvariable imageaddfilename}
\_protected\_edef\_pdfpkresolution        {\_pdfvariable pkresolution}
\_protected\_edef\_pdfinclusioncopyfonts  {\_pdfvariable inclusioncopyfonts}
\_protected\_edef\_pdfinclusionerrorlevel {\_pdfvariable inclusionerrorlevel}
\_protected\_edef\_pdfgentounicode        {\_pdfvariable gentounicode}
\_protected\_edef\_pdfpagebox             {\_pdfvariable pagebox}
\_protected\_edef\_pdfmajorversion        {\_pdfvariable majorversion}
\_protected\_edef\_pdfminorversion        {\_pdfvariable minorversion}
\_protected\_edef\_pdfuniqueresname       {\_pdfvariable uniqueresname}
\_protected\_edef\_pdfhorigin             {\_pdfvariable horigin}
\_protected\_edef\_pdfvorigin             {\_pdfvariable vorigin}
\_protected\_edef\_pdflinkmargin          {\_pdfvariable linkmargin}
\_protected\_edef\_pdfdestmargin          {\_pdfvariable destmargin}
\_protected\_edef\_pdfthreadmargin        {\_pdfvariable threadmargin}
\_protected\_edef\_pdfpagesattr           {\_pdfvariable pagesattr}
\_protected\_edef\_pdfpageattr            {\_pdfvariable pageattr}
\_protected\_edef\_pdfpageresources       {\_pdfvariable pageresources}
\_protected\_edef\_pdfxformattr           {\_pdfvariable xformattr}
\_protected\_edef\_pdfxformresources      {\_pdfvariable xformresources}
\_protected\_edef\_pdfpkmode              {\_pdfvariable pkmode}

\_public
   \pdftexversion \pdftexrevision \pdflastlink \pdfretval \pdflastobj
   \pdflastannot \pdfxformname \pdfcreationdate \pdffontname \pdffontobjnum
   \pdffontsize \pdfpageref \pdfcolorstackinit \pdfliteral \pdfcolorstack
   \pdfsetmatrix \pdfsave \pdfrestore \pdfobj \pdfrefobj \pdfannot
   \pdfstartlink \pdfendlink \pdfoutline \pdfdest \pdfthread \pdfstartthread
   \pdfendthread \pdfinfo \pdfcatalog \pdfnames \pdfincludechars \pdffontattr
   \pdfmapfile \pdfmapline \pdftrailer \pdfglyphtounicode
   \pdfcompresslevel \pdfrunninglinkoff \pdfrunninglinkon
   \pdfobjcompresslevel \pdfdecimaldigits \pdfgamma \pdfimageresolution
   \pdfimageapplygamma \pdfimagegamma \pdfimagehicolor \pdfimageaddfilename
   \pdfpkresolution \pdfinclusioncopyfonts \pdfinclusionerrorlevel
   \pdfgentounicode \pdfpagebox \pdfmajorversion \pdfminorversion \pdfuniqueresname
   \pdfhorigin \pdfvorigin \pdflinkmargin \pdfdestmargin \pdfthreadmargin \pdfpagesattr
   \pdfpageattr \pdfpageresources \pdfxformattr \pdfxformresources \pdfpkmode ;

\_pdfmajorversion     = 1
\_pdfminorversion     = 7 % was 5, see https://tug.org/pipermail/tex-live/2025-February/
\_pdfobjcompresslevel = 2
\_pdfcompresslevel    = 9
\_pdfdecimaldigits    = 3
\_pdfpkresolution     = 600
```

2.4 Basic macros

We define first bundle of basic macros.

basic-macros.opm

```
\_codedecl \sdef {Basic macros for OpTeX <2023-11-11>} % preloaded in format
```

`\bgroup`, `\egroup`, `\empty`, `\space`, and `\null` are classical macros from plain TeX.

basic-macros.opm

```
\_let\_bgroup={  \_let\_egroup=}
\_def \_empty {}
\_def \_space { }
\_def \_null {\_hbox{}}
\_public \bgroup \egroup \empty \space \null ;
```

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

basic-macros.opm

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

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

basic-macros.opm

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

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

basic-macros.opm

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

`\adef` ⟨*char*⟩{⟨*body*⟩} defines active ⟨*char*⟩ as ⟨*body*⟩ and then puts the ⟨*char*⟩ as active character. I.e. the ⟨*body*⟩ can include the ⟨*char*⟩ as non-active charter (if it is non-active before `\adef`). For example `\adef ?{\,?}`. If the character is special, you can escape it, for example `\adef\%{...}`. The space can be declared by `\adef{ }{`⟨*body*⟩`}`. You can declare a macro with parameters too, for example `\adef @#1{...#1...}`. You can use prefixes `\protected`, `\global`, `\long` before `\adef`, they behave like prefixes before `\def`.

basic-macros.opm

```
\_def\_adef#1#2#{\_adefA{#1}{#2}}
\_def\_adefA#1#2#3{\_ea\_def\_directlua{tex.cprint(13,"\_luaescapestring{\_csstring#1}")}#2{#3}%
   \_catcode`#1=13 }
\_public \adef ;
```

`\cs` {⟨*text*⟩} is only a shortcut to `\csname` ⟨*text*⟩`\endcsname`, but you need one more `\_ea` if you need to get the real control sequence \⟨*text*⟩.
`\trycs` {⟨*csname*⟩}{⟨*text*⟩} expands to \⟨*csname*⟩ if it is defined else to the ⟨*text*⟩.

basic-macros.opm

```
\_def \_cs #1{\_csname#1\_endcsname}
\_def \_trycs#1#2{\_ifcsname #1\_endcsname \_csname #1\_ea\_endcsname \_else \_afterfi{#2}\_fi}
\_public \cs \trycs ;
```

`\addto` `\macro{`⟨*text*⟩`}` adds ⟨*text*⟩ to your `\macro`, which must be defined.
`\aheadto` `\macro{`⟨*text*⟩`}` defines `\macro` as ⟨*text*⟩ followed by the original `\macro` body.

basic-macros.opm

```
\_long\_def \_addto #1#2{\_ea\_def\_ea#1\_ea{#1#2}}
\_long\_def \_aheadto #1#2{\_edef#1{\_unexpanded{#2}\_unexpanded\_ea{#1}}}

\_public \addto \aheadto ;
```

`\incr`⟨*counter*⟩ increases ⟨*counter*⟩ by one globally. `\decr`⟨*counter*⟩ decreases ⟨*counter*⟩ by one globally.

basic-macros.opm

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

`\opwarning` {⟨*text*⟩} prints warning on the terminal and to the log file.

basic-macros.opm

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

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

basic-macros.opm

```
\_def\_loggingall{
  \_tracingstats=2 \_tracingpages=1
  \_tracingoutput=1 \_tracingmacros=3 % \_tracinglostchars=2 is already set
  \_tracingparagraphs=1 \_tracingscantokens=1 \_tracingifs=1 \_tracinggroups=1
  \_tracingcommands=3 \_tracingrestores=1 \_tracingassigns=1 }
\_def\_tracingall{\_tracingonline=1 \_loggingall}
\_public \loggingall \tracingall ;
```

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

basic-macros.opm

```
\_def\_banner {This is OpTeX (Olsak's Plain TeX), version <\_optexversion>}%
\_private \optexversion \fmtname ;
```

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

basic-macros.opm

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

2.5 Allocators for TeX registers

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

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

User can use `\dimen0` to `\dimen200` and similarly for `\skip`, `\muskip`, `\box`, and `\toks` directly. User can use `\count20` to `\count200` directly too. This is the same philosophy as in old plainTeX, but the range of directly used registers is wider.

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

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

```
\newcount \_maicount    % redefine maximal allocation index as variable
\_maicount = \maicount  % first value is top of the array

\def\newcountarray #1[#2]{% \newcountarray \foo[100]
    \global\advance\_maicount by -#2\relax
    \ifnum \_countalloc > \_maicount
        \errmessage{No room for a new array of \string\count}%
    \else
        \global\chardef#1=\_maicount
    \fi
}
\def\usecount #1[#2]{%  \usecount \foo[2]
    \count\numexpr#1+#2\relax
}
```

alloc.opm

```
\_codedecl \newdimen {Allocators for registers <2023-02-03>} % preloaded in format
```

The limits are set first.

alloc.opm

```
\_chardef\_maicount = 65535    % Max Allocation Index for counts registers in LuaTeX
\_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
```

Each allocation macro needs its own counter.

alloc.opm

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

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

alloc.opm

```
\_def\_allocator #1#2#3{%
   \_incr{\_cs{_#2alloc}}%
   \_ifnum\_cs{_#2alloc}>\_cs{_mai#2}%
      \_errmessage{No room for a new \_ea\_string\_csname #2\_endcsname}%
   \_else
      \_global#3#1=\_cs{_#2alloc}%
      \_wloga{\_string#1=\_ea\_string\_csname #2\_endcsname\_the\_cs{_#2alloc}}%
   \_fi
}
\_let\_wloga=\_wlog % you can suppress the logging by \_let\_wloga=\_ignoreit
```

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

alloc.opm

```
\_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}

\_public \newcount \newdimen \newskip \newmuskip \newbox \newtoks
         \newread \newwrite \newfam \newlanguage ;
```

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

alloc.opm

```
\_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 ;
```

Other allocation macros `\newmarks`. `\newattribute` and `\newcatcodetable` have their counter allocated by the `\newcount` macro. `\_noattr` is constant `-"7FFFFFFF`, i.e. unused attribute alloc.opm

```
\_newcount \_marksalloc \_marksalloc=0 % start at 1, 0 is \mark
\_chardef\_maimarks=\_maicount
\_def\_newmarks #1{\_allocator #1{marks}\_chardef}

\_newcount \_attributealloc  \_attributealloc=0
\_chardef\_maiattribute=\_numexpr\_maicount -1\_relax
\_attributedef\_noattr \_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.

alloc.opm

```
\_newcount \tmpnum  \_newcount \_tmpnum
\_newdimen \tmpdim  \_newdimen \_tmpdim
```

A few registers: `\maxdimen`, `\hideskip` and `\centering` are initialized like in plainTeX. We absolutely don't support the @category dance, so `\z@skip` `\z@`, `\p@` etc. are defined but not recommended. The `\_zo`, `\_zoskip` and `\voidbox` (equivalents to `\z@`, `\z@skip` and `\voidb@x`) are preferred in OpTeX.

alloc.opm

```
\_newdimen\_maxdimen \_maxdimen=16383.99999pt % the largest legal <dimen>
\_newskip\_hideskip \_hideskip=-1000pt plus 1fill % negative but can grow
\_newskip\_centering \_centering=0pt plus 1000pt minus 1000pt
\_newdimen\_zo \_zo=0pt
\_newskip\_zoskip \_zoskip=0pt plus0pt minus0pt
\_newbox\_voidbox % permanently void box register

\_public \maxdimen \hideskip \centering \voidbox ;
```

2.6 If-macros, loops, is-macros, cases

if-macros.opm

```
\_codedecl \newif {Special if-macros, is-macros and loops <2025-03-01>} % preloaded in format
```

2.6.1 Classical `\newif`

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

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

if-macros.opm

```
\_def\_newif #1{\_ea\_newifA \_string #1\_relax#1}
\_ea\_def \_ea\_newifA \_string\if #1\_relax#2{%
   \_sdef{#1true}{\_let#2=\_iftrue}%
   \_sdef{#1false}{\_let#2=\_iffalse}%
   \_let#2=\_iffalse
}
\_def\_newifi #1{\_ea\_newifiA \string#1\_relax#1}
\_ea\_def \_ea\_newifiA \_string\_if #1\_relax#2{%
   \_sdef{_#1true}{\_let#2=\_iftrue}%
   \_sdef{_#1false}{\_let#2=\_iffalse}%
   \_let#2=\_iffalse
}
\_public \newif ;
```

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

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

`\afterxfi` `{`⟨*what to do*⟩`}`⟨*ignored*⟩`\xfipoint` closes all opened `\if..\fi` conditionals until `\xfipoint` is found and then proceses ⟨*what to do*⟩. The ⟨*ignored*⟩ text must expand to nothing (or spaces only); arbitrary number of opened conditionals can be closed here. Example:

```
\ifodd#1 \ifodd#2 \afterxfi{odd-odd}  \else \afterxfi{odd-even}  \fi
\else    \ifodd#2 \afterxfi{even-odd} \else \afterxfi{even-even} \fi
\fi \xfipoint
```

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

if-macros.opm

```
\_long\_def \_afterfi#1#2\_fi{\_fi#1}
\_long\_def \afterfi#1#2\fi{\_fi#1}
\_let\_xfipoint=\_empty  \_let\xfipoint=\_empty % \xfipoint alone does nothing
\_long\_def \_afterxfi#1#2\_xfipoint{\_romannumeral#20 #1}
\_long\_def \afterxfi#1#2\xfipoint{\_romannumeral#20 #1}
```

2.6.2 Loops

The `\loop` ⟨*codeA*⟩ `\ifsomething` ⟨*codeB*⟩ `\repeat` loops ⟨*codeA*⟩⟨*codeB*⟩ until `\ifsomething` is false. Then ⟨*codeB*⟩ is not executed and loop is finished. This works like in plain TeX, but implementation is somewhat better (you can use `\else` clause after the `\ifsomething`). There are public version `\loop...\repeat` and private version `\_loop` ...`\_repeat`. You cannot mix both versions in one loop.

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

if-macros.opm

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

`\xloop` ⟨*parameters*⟩`\do` ⟨*codeA*⟩ `\ifsomething` ⟨*codeB*⟩ `\repeat` or
`\_xloop` ⟨*parameters*⟩`\_do` ⟨*codeA*⟩ `\_ifsomething` ⟨*codeB*⟩ `\_repeat` behave analogically like `\loop`, but moreover: you can specify a rule of parameters scanning (like in `\def` primitive) and you can use these parameters (i.e. `#1`, `#2`, etc.) in the ⟨*codeA*⟩ or ⟨*codeB*⟩. Each iteration of this loop reads next parameters from the input queue (just after `\repeat`) using the ⟨*parameters*⟩ rule. Example:

```
\def\countparams{\tmpnum=0
   \xloop ##1,\do \ifx\end##1\empty \else \incr\tmpnum \repeat}
\def\list{A, B, C}
\ea \countparams \list,\end, % \tmpnum = the number of comma separated parameters
```

Second new feature: the `\xloop` macro is expandable, if ⟨*codeA*⟩ and ⟨*codeB*⟩ include only expandable macros. This is not the case of previous example but you can prefix `\tmpnum=0` and `\incr` by the `\immediateassignment` LuaTeX primitive and you get expandable macro `\countparams`.

if-macros.opm

```
\_long\_def\xloop#1\do#2\repeat{\_immediateassignment\_long\_def\_body#1{#2\_ea\_body\_fi}\_body}
\_long\_def\_xloop#1\_do#2\_repeat{\_immediateassignment\_long\_def\_body#1{#2\_ea\_body\_fi}\_body}
```

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

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

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

Recommendation: it is better to use private variants of `\_foreach`. When the user writes `\load[tikz]` then `\foreach` macro is redefined in each TikZ environment. The private variants use `\_do` separator instead `\do` separator and it isn't redefined.
`\foreachx` `\list` behaves like `\ea \foreach \list` but the `\list` is expanded later, when its data are actually needed. This is more effective becuase the macro needn't skip a potentially huge amount of data in order to scan `\do{`⟨*what*⟩`}`.

if-macros.opm

```
\_newcount\_frnum        % the numeric variable used in \fornum
\_def\_do{\_doundefined} % we need to ask \_ifx#1\_do ...

\_def\_foreach{\_foreachx\_empty}  \_def\foreach{\_foreachAx\_empty}
\_long\_def\_foreachx #1\_do #2#{\_foreachAz{#1}{#2}}
\_long\_def\_foreachAx #1\do #2#{\_foreachAz{#1}{#2}} \_let\foreachx=\_foreachAx
\_long\_def\_foreachAz #1#2{\_isempty{#2}\_iftrue
   \_afterfi{\_foreachA{#1}{##1}}\_else\_afterfi{\_foreachA{#1}{#2}}\_fi}
\_long\_def\_foreachA #1#2#3{\_putforstack
   \_immediateassignment \_long\_gdef\_fbody#2{\_testparam##1..\_iftrue #3\_ea\_fbody\_fi}%
   \_ea\_fbody #1#2\_finbody\_getforstack
}
\_long\_def\_testparam#1#2#3\_iftrue{\_ifx###1\_empty\_ea\_finbody\_else}
\_long\_def\_finbody#1\_finbody{}
```

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

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

if-macros.opm

```
\_def\_fornum#1..#2\_do{\_fornumstep 1:#1..#2\_do}
\_long\_def\_fornumstep#1:#2..#3\_do#4{\_putforstack
   \_immediateassigned{%
      \_gdef\_fbody##1{#4}%
      \_global\_frnum=\_numexpr#2\_relax
   }%
   \_ea\_fornumB\_ea{\_the\_numexpr#3\_ea}\_ea{\_the\_numexpr#1}%
}
\_def\_fornumB #1#2{\_ifnum#1\_ifnum#2>0<\_else>\_fi \_frnum \_getforstack
   \_else \_afterfi{\_ea\_fbody\_ea{\_the\_frnum}%
      \_immediateassignment\_global\_advance\_frnum by#2
      \_fornumB{#1}{#2}}\_fi
}
\_def\fornum#1..#2\do{\_fornumstep 1:#1..#2\_do}
\_def\fornumstep#1:#2..#3\do{\_fornumstep #1:#2..#3\_do}
```

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

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

Implementation note: we cannot use TeX-groups for nesting levels because we want to do the macros expandable. We must implement a special for-stack which saves the data needed by `\foreach` and `\fornum`. The **`\_putforstack`** is used when `\for*` is initialized and **`\_getforstack`** is used when the `\for*` macro ends. The **`\_forlevel`** variable keeps the current nesting level. If it is zero, then we need not save nor restore any data.

if-macros.opm

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

User can define own expandable “foreach” macro by `\foreachdef` `\macro` ⟨*parameter-mask*⟩`{`⟨*what*⟩`}` which can be used by `\macro` `{`⟨*list*⟩`}`. The macro reads repeatedly parameters from ⟨*list*⟩ using ⟨*parameter-mask*⟩ and does ⟨*what*⟩ for each such reading. For example

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

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

if-macros.opm

```
\_def\_foreachdef#1#2#{\_toks0{#2}%
   \_long\_edef#1##1{\_ea\_noexpand\_csname _body:\_csstring#1\_endcsname
                     ##1\_the\_toks0 \_noexpand\_finbody}%
   \_foreachdefA#1{#2}}
\_long\_def\_foreachdefA#1#2#3{%
   \_long\_sdef{_body:\_csstring#1}#2{\_testparam##1..\_iftrue #3\_cs{_body:\_csstring#1\_ea}\_fi}}

\_public \foreachdef ;
```

2.6.3 Is-macros and selection of cases

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

`\issomething` ⟨*params*⟩ `\iftrue` ⟨*codeA*⟩ `\else` ⟨*codeB*⟩ `\fi`
`or`
`\issomething` ⟨*params*⟩ `\iffalse` ⟨*codeB*⟩ `\else` ⟨*codeA*⟩ `\fi`

The `\else` part is optional. The ⟨*codeA*⟩ is processed if `\issomething`⟨*params*⟩ generates true condition. The ⟨*codeB*⟩ is processed if `\issomething`⟨*params*⟩ generates false condition.

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

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

`\isempty` `{`⟨*text*⟩`}\iftrue` is true if the ⟨*text*⟩ is empty. This macro is expandable.
`\istoksempty` ⟨*tokens variable*⟩`\iftrue` is true if the ⟨*tokens variable*⟩ is empty. It is expandable.

if-macros.opm

```
\_long\_def \_isempty #1#2{\_if\_relax\_detokenize{#1}\_relax \_else \_ea\_unless \_fi#2}
\_def \_istoksempty #1#2{\_ea\_isempty\_ea{\_the#1}#2}
\_public \isempty \istoksempty ;
```

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

if-macros.opm

```
\_long\_def\_isequal#1#2#3{\_directlua{%
   if "\_luaescapestring{\_detokenize{#1}}"=="\_luaescapestring{\_detokenize{#2}}"
   then else tex.print("\_nbb unless") end}#3}
\_public \isequal ;
```

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

if-macros.opm

```
\_long\_def\_ismacro#1{\_ea\_isequal\_ea{#1}}
\_public \ismacro ;
```

`\isdefined` `{`⟨*csname*⟩`}\iftrue` is true if `\`⟨*csname*⟩ is defined. The macro is expandable.

if-macros.opm

```
\_def\_isdefined #1#2{\_ifcsname #1\_endcsname \_else \_ea\_unless \_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.

if-macros.opm

```
\_long\_def\_isinlist#1#2#3{%
   \_immediateassignment\_long\_def\_isinlistA##1#2##2\_end/_%
        {\_if\_relax\_detokenize{##2}\_relax \_ea\_unless\_fi#3}%
   \_ea\_isinlistA#1\_endlistsep#2\_end/_%
}
\_public \isinlist ;
```

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

if-macros.opm

```
\_newread \_testin
\_def\_isfile #1#2{%
   \_openin\_testin ={#1}\_relax
   \_ifeof\_testin \_ea\_unless
   \_else \_closein\_testin
   \_fi #2%
}
\_public \isfile ;
```

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

if-macros.opm

```
\_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.

if-macros.opm

```
\_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⟩}
  ...
  ⟨token-n⟩  {⟨what to do if token=token-n⟩}
  \_finc     {⟨what to do in other cases⟩}
```

The meaning of tokens are compared by `\ifx` primitive. The parts ⟨*what to do*⟩ can be finalized by a macro which can read more data from the input stream as its parameters.

if-macros.opm

```
\_long\_def \_casesof #1#2#3{\_ifx #2\_finc \_ea\_ignoresecond \_else \_ea\_usesecond \_fi
   {#3}{\_ifx#1#2\_ea\_ignoresecond \_else \_ea\_usesecond \_fi {\_finc{#3}}{\_casesof#1}}%
}
\_long\_def \_finc #1#2\_finc#3{#1}

\_public \casesof ;
```

`\qcasesof` `{`⟨*string*⟩`}` ⟨*list of cases*⟩ behaves like `\casesof` but it compares phrases with the given ⟨*string*⟩ using `\isequal`. The ⟨*list of cases*⟩ includes pairs `{`⟨*phrases*⟩`} {`⟨*what to do if string=phrase*⟩`}` finalized by a pair `\_finc {`⟨*what to do else*⟩`}`. The ⟨*phrases*⟩ is a single phrase or phrases separated by `|` which means "or". For example the pair `{ab|cde|f} {`⟨*code*⟩`}` runs ⟨*code*⟩ if the given ⟨*string*⟩ is `ab` or `cde` or `f`. The usage of `\qcasesof` can be found in OpTeX trick 0132.

if-macros.opm

```
\_long\_def \_qcasesof #1#2#3{\_ifx\_finc#2\_ea\_ignoresecond \_else \_ea\_usesecond \_fi
   {#3}{\_qcasesofA{#1}#2|\_qcasesofA|{\_finc{#3}}{\_qcasesof{#1}}}%
}
\_long\_def\_qcasesofA#1#2|{\_ifx\_qcasesofA#2\_ea\_usesecond \_else
   \_isequal{#1}{#2}\_iftrue \_qcasesofB \_fi \_afterfi{\_qcasesofA{#1}}\_fi
}
\_long\_def\_qcasesofB #1\_qcasesofA|#2#3{\_fi\_fi#2}

\_public \qcasesof ;
```

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

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

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

if-macros.opm

```
\_def \_xcasesof {\_nospacefuturelet\_next\_xcasesofA}
\_def \_xcasesofA {\_ifx\_next\_finc \_ea\_usesecond \_else \_ea \_xcasesofB \_fi}
\_long\_def \_xcasesofB #1#2{%
   #1\_ea\_ignoresecond\_else \_ea\_usesecond\_fi {\_finc{#2}}{\_xcasesof}%
}
\_public \xcasesof ;
```

2.7 Setting parameters

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

- primitive registers used in built-in algorithms of TeX,
- registers declared and used by OpTeX macros.

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

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

parameters.opm

```
\_codedecl \normalbaselineskip {Parameter settings <2023-09-19>} % preloaded in format
```

2.7.1 Primitive registers

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

parameters.opm

```
\_parindent=20pt     % indentation of paragraphs
\_pretolerance=100   % parameters used in paragraph breaking algorithm
\_tolerance=200
\_hbadness=1000
\_vbadness=1000
\_doublehyphendemerits=10000
\_finalhyphendemerits=5000
\_adjdemerits=10000
\_uchyph=1
\_defaulthyphenchar=`
\_defaultskewchar=-1
\_hfuzz=0.1pt
\_vfuzz=0.1pt
\_overfullrule=5pt
```

```
\_linepenalty=10     % penalty between lines inside the paragraph
\_hyphenpenalty=50   % when a word is bro-ken
\_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
\_predisplaypenalty=10000  % above display math
\_postdisplaypenalty=0     % below display math
\_delimiterfactor=901 % parameter for scaling delimiters
\_delimitershortfall=5pt
\_nulldelimiterspace=1.2pt
%\_scriptspace=0.5pt % \Umathspaceafterscript 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`).

parameters.opm

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

The following two registers were introduced to fix a couple of bugs in the LuaTEX engine. When `\matheqdirmode` is positive short skip detection around display equations will work with right to left typesetting. When `\breakafterdirmode` is set to 1 a glue after a dir node will not be ignored.

parameters.opm

```
\_ifx\_matheqdirmode\_undefined \_else
\_matheqdirmode=1
\_breakafterdirmode=1
\_fi
```

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`.

parameters.opm

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

\_public \smallskipamount \medskipamount \bigskipamount
   \normalbaselineskip \normallineskip \normallineskiplimit
   \jot \interdisplaylinepenalty \interfootnotelinepenalty ;
```

Plain TEX macros for setting parameters. `\normalbaselines`, `\frenchspacing`, `\nonfrenchspacing`.

parameters.opm

```
\_def\_normalbaselines{\_lineskip=\_normallineskip
  \_baselineskip=\_normalbaselineskip \_lineskiplimit=\_normallineskiplimit}

\_def\_frenchspacing{\_sfcode`\.=1000 \_sfcode`=1000 \_sfcode`=1000
```

```
  \_sfcode`=1000 \_sfcode`=1000 \_sfcode`=1000 }
\_def\_nonfrenchspacing{\_sfcode`\.=3000 \_sfcode`=3000 \_sfcode`=3000
  \_sfcode`=2000 \_sfcode`=1500 \_sfcode`=1250 }

\_public \normalbaselines \frenchspacing \nonfrenchspacing ;
```

2.7.3 Different settings than in plain TEX

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

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

parameters.opm

```
\_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 1 in, 1 in coordinates) and set default page dimensions as A4, not letter.

parameters.opm

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

\_clubpenalty=10000    % after first line of paragraph
\_widowpenalty=10000   % before last line of paragraph

\_showboxbreadth=150   % for tracing boxes
\_showboxdepth=7
\_errorcontextlines=15
\_tracinglostchars=2   % missing character warnings on terminal too

\_outputmode=1   % PDF output
\_pdfvorigin=0pt % origin is exactly at upper left corner
\_pdfhorigin=0pt
\_hoffset=25mm   % margins are 2.5cm, no 1in
\_voffset=25mm
\_hsize=160mm    % 210mm (from A4 size) - 2*25mm (default margins)
\_vsize=244mm    % 297mm (from A4 size) - 2*25mm (default margins) -3mm baseline correction
\_pdfpagewidth=210 true mm
\_pdfpageheight=297 true mm
```

If you insist on plain TEX values of these parameters then you can call the `\plaintexsetting` macro.

parameters.opm

```
\_def\_plaintexsetting{%
   \_emergencystretch=0pt
   \_clubpenalty=150
   \_widowpenalty=150
   \_pdfvorigin=1in
   \_pdfhorigin=1in
   \_hoffset=0pt
   \_voffset=0pt
   \_hsize=6.5in
   \_vsize=8.9in
   \_pdfpagewidth=8.5 true in
   \_pdfpageheight=11 true in
   \_nonfrenchspacing
}
\_public \plaintexsetting ;
```

2.7.4 OpTEX parameters

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

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

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

`parameters.opm`

```
\_newtoks\_picdir
\_public \picdir ;
```

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={page3}` selects page 3 from included PDF file.

`parameters.opm`

```
\_newdimen\_picwidth   \_picwidth=0pt   \_let\picw=\_picwidth
\_newdimen\_picheight  \_picheight=0pt
\_newtoks\_picparams
\_public \picwidth \picheight \picparams ;
```

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

`parameters.opm`

```
\_newtoks \_kvdict
\_public \kvdict ;
```

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.

`parameters.opm`

```
\_newtoks\_everytt
\_newtoks\_everyintt
\_public \everytt \everyintt ;
```

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.

`parameters.opm`

```
\_newcount\_ttline    \_ttline=-1  % last line number in \begtt...\endtt
\_public \ttline ;
```

The `\ttindent` gives default indentation of verbatim lines printed by `\begtt`...`\endtt` pair or by `\verbinput`.
The `\ttshift` gives the amount of shift of all verbatim lines to the right. Despite the `\ttindent`, it does not shift the line numbers, only the text.
The `\iindent` gives default indentations used in the table of contents, captions, lists, bib references,
It is strongly recommended to re-set this value if you set `\parindent` to another value than plain TeX

default 20pt. A well-typeset document should have the same dimension for all indentations, so you should say `\ttindent=\parindent` and `\iindent=\parindent`.

parameters.opm

```
\_newdimen\_ttindent \_ttindent=\_parindent % indentation in verbatim
\_newdimen\_ttshift
\_newdimen\_iindent  \_iindent=\_parindent
\_public \ttindent \ttshift \iindent ;
```

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

parameters.opm

```
\_newcount \_tabspaces   \_tabspaces=3
\_public \tabspaces ;
```

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

parameters.opm

```
\_newtoks\_hicolors
\_public \hicolors ;
```

The default item mark used between `\begitems` and `\enditems` is the bullet. The `\defaultitem` tokens list declares this default item mark.
The `\everyitem` tokens list is applied in vertical mode at the start of each item.
The `\everylist` tokens list is applied after the group is opened by `\begitems`
The `\ilevel` keeps the value of the current nesting level of the items list.
The `\olistskipamount` is vertical skip above and below the items list if `\ilevel=1`.
The `\ilistskipamount` is vertical skip above and below the items list if `\ilevel>1`.
The `\itemskipamount` is vertical skip between list items, but not above the first and below the last.

parameters.opm

```
\_newtoks\_defaultitem     \_defaultitem={$\_bullet$\_enspace}
\_newtoks\_everyitem
\_newtoks\_everylist
\_newcount \_ilevel
\_newskip\_olistskipamount \_olistskipamount=\_medskipamount
\_newskip\_ilistskipamount \_ilistskipamount=0pt plus.5\_smallskipamount
\_newskip\_itemskipamount  \_itemskipamount=0pt

\_public \defaultitem \everyitem \everylist \ilevel
         \olistskipamount \ilistskipamount \itemskipamount ;
\_let \listskipamount = \_olistskipamount  % for backward compatibility
```

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

parameters.opm

```
\_newskip\_titskip   \_titskip=40pt \_relax  % \vglue above title printed by \tit
\_public \titskip ;
```

The `\begmulti` and `\endmulti` pair creates more columns. The parameter `\colsep` declares the space between columns. If n columns are specified then we have $n-1$ `\colseps` and n columns in total `\hsize`. This gives the definite result of the width of the columns.

parameters.opm

```
\_newdimen\_colsep \_colsep=20pt  % space between columns
\_public \colsep ;
```

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

parameters.opm

```
\_newtoks \_everytocline
\_public \everytocline ;
```

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 2.32.6.
The `\bibpart` saves the name of bib-list if there are more bib-lists in single document, see section 2.32.1.

parameters.opm

```
\_newtoks\_bibtexhook
\_newtoks\_biboptions
\_newtoks\_bibpart
\_public \bibtexhook \biboptions \bibpart ;
```

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

parameters.opm

```
\_newtoks\_everycaptiont  \_newtoks\_everycaptionf
\_public \everycaptiont \everycaptionf ;
```

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

parameters.opm

```
\_newtoks\_everyii
\_public \everyii ;
```

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

parameters.opm

```
\_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 ;
```

The `\table` parameters follow. The `\thistable` tokens list register should be used for giving an exception for only one `\table` which follows. It should change locally other parameters of the `\table`. It is reset to an empty list after the table is printed.
The `\everytable` tokens list register is applied in every table. There is another difference between these two registers. The `\thistable` is used first, then strut and baselineskip settings are done, then `\everytable` is applied and then the table is printed.
`\tabstrut` configures the height and depth of lines in the table. You can declare `\tabstrut={}`, then normal baselineskip is used in the table. This can be used when you don't use horizontal nor vertical lines in tables.
`\tabiteml` is applied before each item, `\tabitemr` is applied after each item of the table.
`\tablinespace` is additional vertical space between horizontal rules and the lines of the table.
`\hhkern` gives the space between horizontal lines if they are doubled and `\vvkern` gives the space between such vertical lines.
`\tabskipl` is `\tabskip` used before first column, `\tabskipr` is `\tabskip` used after the last column.
`\tsize` is virtual unit of the width of paragraph-like table items when `\table pxto`⟨*size*⟩ is used.

parameters.opm

```
\_newtoks\_everytable \_newtoks\_thistable
\_newtoks\_tabiteml \_newtoks\_tabitemr \_newtoks\_tabstrut
\_newdimen\_tablinespace \_newdimen\_vvkern \_newdimen\_hhkern \_newdimen\_tsize
\_newskip\_tabskipl  \_newskip\_tabskipr
\_everytable={}        % code used after settings in \vbox before table processing
\_thistable={}         % code used when \vbox starts, is is removed after using it
\_tabstrut={\_strut}
\_tabiteml={\_enspace} % left material in each column
\_tabitemr={\_enspace} % right material in each column
\_tablinespace=2pt     % additional vertical space before/after horizontal rules
\_vvkern=1pt           % space between double vertical line and used in \frame
\_hhkern=1pt           % space between double horizontal line and used in \frame
\_tabskipl=0pt\_relax  % \tabskip used before first column
\_tabskipr=0pt\_relax  % \tabskip used after the last column
\_public \everytable \thistable \tabiteml \tabitemr \tabstrut \tablinespace
         \vvkern \hhkern \tsize \tabskipl \tabskipr ;
```

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

parameters.opm

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

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

parameters.opm

```
\_newtoks \_lmfil   \_lmfil={\_hfil}
\_public \lmfil ;
```

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

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

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

parameters.opm

```
\_newtoks\_headline   \_headline={}
\_newtoks\_footline   \_footline={\_hss\_rmfixed \_numprint\_folio \_hss}
\_public \headline \footline ;
```

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

parameters.opm

```
\_newdimen \_headlinedist  \_headlinedist=14pt
\_newdimen \_footlinedist  \_footlinedist=24pt
\_public \headlinedist \footlinedist ;
```

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

parameters.opm

```
\_newskip \_pgbottomskip  \_pgbottomskip=0pt \_relax
\_public \pgbottomskip ;
```

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:

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

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

parameters.opm

```
\_newtoks \_nextpages
\_public \nextpages ;
```

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:

```
\pgbackground={\Yellow \hrule height 0pt depth\pdfpageheight width\pdfpagewidth}
```

parameters.opm

```
\_newtoks \_pgbackground   \_pgbackground={} % for page background
\_public \pgbackground   ;
```

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.

parameters.opm

```
\_newtoks \_ovalparams
\_newtoks \_circleparams
%\_ovalparams={\_roundness=2pt \_fcolor=\Yellow \_lcolor=\Red \_lwidth=.5bp
%              \_shadow=N \_overlapmargins=N \_hhkern=0pt \_vvkern=0pt }
%\_circleparams={\_ratio=1 \_fcolor=\Yellow \_lcolor=\Red \_lwidth=.5bp
%                \_shadow=N \_overlapmargins=N \_hhkern=3pt \_vvkern=3pt}

\_newdimen \_roundness    \_roundness=5mm % used in \clippingoval macro
\_public \ovalparams \circleparams \roundness ;
```

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

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

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

parameters.opm

```
\_let\cnvinfo=\_ignoreit
```

2.8 More OpTeX macros

The second bundle of OpTeX macros is here.

more-macros.opm

```
\_codedecl \eoldef {OpTeX useful macros <2024-10-31>} % preloaded in format
```

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

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

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

more-macros.opm

```
\_def\_opinput #1{\_setctable\_optexcatcodes
   \_savecatcodetable\_tmpcatcodes \_catcodetable\_tmpcatcodes
   \_input {#1}\_relax\_restorectable}

\_newcatcodetable \_optexcatcodes
\_newcatcodetable \_plaintexcatcodes
\_newcatcodetable \_tmpcatcodes

\_public \optexcatcodes \plaintexcatcodes \opinput ;

\_savecatcodetable\_optexcatcodes
{\_catcode`_=8 \savecatcodetable\plaintexcatcodes}
```

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.

more-macros.opm

```
\_catcodetable0

\_def\_setctable#1{\_edef\_ctablelist{{\_the\_catcodetable}\_ctablelist}%
   \_catcodetable#1\_relax
}
\_def\_restorectable{\_ea\_restorectableA\_ctablelist\_relax}
\_def\_restorectableA#1#2\_relax{%
   \_ifx^#2^\_opwarning
      {You can't use \_noindent\restorectable without previous \_string\setctable}%
   \_else \_def\_ctablelist{#2}\_catcodetable#1\_relax \_fi
}
\_def\_ctablelist{.}

\_public \setctable \restorectable ;
```

When a special macro is defined with different catcodes then `\normalcatcodes` can be used at the end of such definition. The normal catcodes are restored. The macro reads catcodes from `\optecatodes` table and sets it to the main catcode table 0.

more-macros.opm

```
\_def\_normalcatcodes {\_catcodetable\_optexcatcodes \_savecatcodetable0 \_catcodetable0 }
\_public \normalcatcodes ;
```

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`.

more-macros.opm

```
\_def \_load [#1]{\_savemathsb \_loadA #1,,,\_end \_restoremathsb}
\_def \_loadA #1#2,{\_ifx,#1 \_ea \_loadE \_else \_loadB{#1#2}\_ea\_loadA\_fi}
\_def \_loadB #1{%
   \_ifcsname _load:#1\_endcsname \_else
      \_isfile {#1.opm}\_iftrue \_opinput {#1.opm}\_else \_opinput {#1}\_fi
      \_sxdef{_load:#1}{}%
      \_trycs{_afterload}{}\_let\_afterload=\_undefined
   \_fi
}
\_def \_loadE #1\_end{}
\_public \load ;
```

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

The `\nospaceafter` ignores the following optional space at expand processor level using the negative `\romannumeral` trick. The `\nospacefuturelet` bahaves like `\futurelet` primitive, but it ignores the following optional space and works at expand processor level.

more-macros.opm

```
\_newtoks\_opt
\_def\_optdef#1[#2]{%
   \_def#1{\_isnextchar[{\_cs{_oA:\_csstring#1}}{\_cs{_oA:\_csstring#1}[#2]}}%
   \_sdef{_oA:\_csstring#1}[##1]{%
      \_immediateassignment\_opt={##1}\_cs{_oB:\_csstring#1\_nospaceafter}}%
   \_sdef{_oB:\_csstring#1\_nospaceafter}%
}
\_def\_nospaceafter#1{\_ea#1\_romannumeral-`\.\_noexpand}
\_def\_nospacefuturelet#1#2{\_ea\_immediateassignment
   \_ea\_futurelet\_ea#1\_ea#2\_romannumeral-`\.\_noexpand}

\_public \opt \optdef \nospaceafter \nospacefuturelet ;
```

`\_noprefix` ⟨*cs*⟩ works like `\csstring` ⟨*cs*⟩, but ignores not only the first backlash but the second "`_`" ignores too (if it follows the backslash).

more-macros.opm

```
\_def\_noprefix#1{\_ea\_noprefixA \_csstring#1\_empty\_fin}
\_def\_noprefixA #1#2\_fin{\_if _#1\_else #1\_fi #2}
```

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

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

The `\skiptoeol` macro reads the text to the end of the current line and ignores it.

more-macros.opm

```
\_def\_eoldef #1{\_def #1{\_begingroup \_catcode``^M=12 \_eoldefA #1}%
   \_ea\_def\_csname _eol:\_noprefix #1\_endcsname}
\_catcode``^M=12 %
\_def\_eoldefA #1#2^^M{\_endgroup\_csname _eol:\_noprefix #1\_endcsname{#2}}%
\_normalcatcodes %

\_eoldef\_skiptoeol#1{}

\_def\_bracedparam#1{%
   \_trycs{_eol:\_noprefix#1}%
      {\_errmessage{\_string\bracedparam: \_string#1 isn't defined by \_string\eoldef}}%
}
\_public \eoldef \skiptoeol \bracedparam ;
```

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

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

more-macros.opm

```
\_def\_scantoeol#1{\_begingroup \_setscancatcodes \_scantoeolA #1}
\_def\_setscancatcodes{\_setverb \_catcode``^M=12\_catcode`=7\_catcode`=10\_catcode``^J=14 }
\_catcode``^M=12 %
\_def\_scantoeolA#1#2^^M{\_endgroup #1{#2}}%
\_normalcatcodes %

\_public \scantoeol ;
```

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

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

more-macros.opm

```
\_newtoks\_repltoks
\_catcode`!=3 \_catcode`?=3
\_long\_def\_replstring #1#2#3{%  \replstring #1{stringA}{stringB}
   \_long\_def\_replacestringsA##1#2{\_repltoks\_ea{##1}\_replacestringsB\_empty}%
   \_long\_def\_replacestringsB##1#2{\_ea\_replacestringsC\_ea{##1}}%
   \_long\_def\_replacestringsC##1{\_ifx!##1\_empty\_else
      \_toksapp\_repltoks{#3##1}\_ea\_replacestringsB\_ea\_empty\_fi}%
   \_ea\_replacestringsA \_ea\_empty#1?#2!#2%
   \_ea\_replacestringsD \_ea\_empty\_the\_repltoks#1}
\_long\_def\_replacestringsD #1?#2{\_repltoks\_ea{#1}\_edef#2{\_the\_repltoks}}%
\_normalcatcodes

\_public \replstring ;
```

You can find three special tricks in the `\replstring` code:

- Adding and removing `?` character: without this the `\def\b{xxxA}\replstring\b{AA}{BB}` will cause an error.
- Using `\_toksapp` instead `\addto`: without this you cannot have `#`$_6$ in the string. Moreover, it significantly reduces the calculation time. Unfortunately, it does `\replstring` unexpandable.
- Using `\_empty` before scanning separated parameters and removing it after scanning by `\ea`: without this TeX removes braces from ⟨*textA*⟩`{xxx}`⟨*textA*⟩.

The `\replstring` macro isn't expandable. But you can see OpTeX trick 0136 and OpTeX trick 0137. These triks implement expandable alternatives and enable more general modifications of macros by regular expressions.

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`.

more-macros.opm

```
\_def\catcode#1{\_catcode \_if`\_noexpand#1\_ea`\_else\_if"\_noexpand#1"\_else
   \_if'\_noexpand#1'\_else \_ea\_ea\_ea\_ea\_ea\_ea\_ea#1\_fi\_fi\_fi}
```

The `\removespaces` ⟨*text with spaces* ⟩`{}` expands to ⟨*textwithoutspaces*⟩.
The `\_ea\ignorept\the`⟨*dimen*⟩ expands to a decimal number `\the`⟨*dimen*⟩ but without `pt` unit.

more-macros.opm

```
\_def\_removespaces #1 {\_isempty{#1}\_iffalse #1\_ea\_removespaces\_fi}
\_ea\_def \_ea\_ignorept \_ea#\_ea1\_detokenize{pt}{#1}

\_public \removespaces \ignorept ;
```

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`⟨*sequence*⟩ solves this problem. If the sequence is equal to a character then it expands to this character (always with catcode 12). If it isn't equal to a character then it expands to nothing. You can say ``\expanded{`\cstochar\foo}`` if you want to extract the character code.

more-macros.opm

```
\_def\_cstochar#1{\_ea\_cstocharA\_meaning#1 {} {} \_fin}
\_def\_cstocharA#1 #2 #3 #4\_fin{\_isinlist{#1#2}-\_iffalse #3\_fi}

\_public \cstochar ;
```

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

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

You can use expandable `\expr{`⟨*expression*⟩`}` for analogical purposes. It expands to the value of the ⟨*expression*⟩ at expand processor level. The ⟨*expression*⟩ can include `+-*/^()` and decimal numbers in common syntax. Moreover, `a//b` means integer division and `a\%b` is remainder. 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 for macro programmers because a user can re-define the control sequences `\expr` or `\bp`.

more-macros.opm

```
\_def\_decdigits{3} % digits after decimal point in \_bp and \_expr outputs.
\_def\_pttopb#1{%
   \_directlua{tex.print(string.format('\_pcent.#1f',
                token.scan_dimen()/65781.76))}%  pt to bp conversion
}
\_def\_bp{\_isnextchar[{\_bpA}{\_bpA[\_decdigits]}}
\_def\_bpA[#1]#2{\_pttopb{#1}\_dimexpr#2\_relax}
\_def\_expr{\_isnextchar[{\_exprA}{\_exprA[\_decdigits]}}
\_def\_exprA[#1]#2{\_directlua{tex.print(string.format('\_pcent.#1f',#2))}}

\_public \expr \bp ;
```

The `\expr` and `\bp` macros return their results with given number of decimal digits even if there are trailing zeros. There is the `\nnum` macro to "normalize" such decimal numbers. `\nnum{`⟨*number*⟩`}` expands its parameter and removes trailing zeros after decimal point and removes the decimal point if nothing follows. For example, use `\nnum{\expr[10]{`⟨*expression*⟩`}}`. The `\nnum` macro is fully expandable.

more-macros.opm

```
\_def\_nnum #1{\_ea\_nnumA\_expanded{#1}.\_fin}
\_def\_nnumA #1.#2\_fin{#1\_ifx~#2~\_else \_nnumB #20.\_fin \_fi}
\_def\_nnumB #10.#2\_fin{\_ifx~#2~\_nnumC#1\_else \_nnumB #1.0.\_fin \_fi}
\_def\_nnumC #1.{\_ifx~#1~\_else .#1\_fi}
\_public \nnum ;
```

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*⟩`}`. The `\_Xpos` macro defines `\_pos:`⟨*label*⟩ as `{`⟨*x-pos*⟩`}{`⟨*y-pos*⟩`}{`⟨*total-pg*⟩`}{`⟨*rel-pg*⟩`}`. We need to read only given parameter by `\_posi`, `\_posii` or `\_posiii` auxiliary macros. The implementation of `\setpos`, `\posx` and `\posy` macros are based on `\padsavepos` `\pdflastxpos` and `\pdflastypos` pdfTeX primitives. The `\pospg` simply reads the data from the `\_currpage` macro.

more-macros.opm

```
\_def\_Xpos#1#2#3{\_sxdef{_pos:#1}{{#2}{#3}\_currpage}}
\_def\_setpos[#1]{\_openref\_pdfsavepos
   \_ewref\_Xpos{{#1}\_unexpanded{{\_the\_pdflastxpos}{\_the\_pdflastypos}}}}

\_def\_posx [#1]{\_ea \_posi   \_expanded {\_trycs{_pos:#1}{{0}{}{}{}}sp}}
\_def\_posy [#1]{\_ea \_posii  \_expanded {\_trycs{_pos:#1}{{}{0}{}{}}sp}}
\_def\_pospg[#1]{\_ea \_posiii \_expanded {\_trycs{_pos:#1}{{}{}{0}{}}}}

\_def\_posi #1#2#3#4{#1}  \_def\_posii #1#2#3#4{#2}  \_def\_posiii #1#2#3#4{#3}

\_public \setpos \posx \posy \pospg ;
```

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

`\_doc` ⟨*ignored text*⟩
⟨*documentation*⟩
`\_cod` ⟨*ignored text*⟩

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

more-macros.opm

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

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

more-macros.opm

```
\_def\_docgen #1 {\_ea \_docgenA \_input{#1.opm}}
\_long \_def\_docgenA #1\_codedecl#2\_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}\vvkern=\kv{margins}\relax
   \kv{frame-color}\frame{\kv{text-color}\strut #1}%
   \egroup
}
```

We recommend using `\optdef` for defining macros with optional parameters written in `[]`. Then the optional parameters are saved in the `\opt` tokens register. First: we read default parameters by `\readkv\myframedefaults` and secondly the actual parameters are read by `\readkv{\the\opt}`. The last setting wins. Third: the values can be used by the expandable `\kv{`⟨*key*⟩`}` macro. The `\kv{`⟨*key*⟩`}` returns `???` if such a ⟨*key*⟩ isn't declared but if `.notdef` key is declared then its value is returned in this case.

The `\kv{`⟨*key*⟩`}` macro fully expands the value of key. If you don't want to expand the value, use `\trykv{`⟨*key*⟩`}{`⟨*code*⟩`}`. It returns unexpanded value of ⟨*key*⟩ if declared else returns expanded ⟨*code*⟩. For example `\edef\macro{\trykv{key}{}}` defines `\macro` as the unexpanded value of the `key`.

You can use keys without values in the parameters list too. Then you can ask if the key is declared by `\iskv{`⟨*key*⟩`}\iftrue` or the key is undeclared by `\iskv{`⟨*key*⟩`}\iffalse`. For example, you write to your documentation of your code that user can set the `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{`⟨*key*⟩`}{`⟨*code*⟩`}` to declare ⟨*code*⟩ which is processed immediately when the ⟨*key*⟩ is processed by `\readkv`. For example

```
\newcount\mydraftmode
\kvx{draft}{\mydraftmode=1 }
\kvx{final}{\mydraftmode=0 }
\optdef\myframe [] #1{...
   \readkv\myframedefaults \readkv{\the\opt}%
   \ifnum\mydraftmode=1 ...draft mode... \else ...final mode... \fi
   ...}
```

The syntax of `\kvx` `{`⟨*key*⟩`}{`⟨*code*⟩`}` allows to use `#1` inside the code. It is replaced by the actual ⟨*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`, `\trykv`, `\iskv`, `\kvx` and `\nokvx` macros use this named dictionary of ⟨*key*⟩/⟨*value*⟩ pairs. Package options can be processed when `\kvdict={pkg:`⟨*pkg*⟩`}`, example is the `\mathset` macro in `math.opm` package.

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

A more extensive example can be found in OpTEX trick 0073.

keyval.opm
```
\_codedecl \readkv {Key-value dictionaries <2023-11-24>} % preloaded in format
```

Implementation.
The `\readkv`⟨*list*⟩ expands its parameter and does replace-strings in order to remove spaces around equal signs and 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*⟩`#1` macro and `\nokvx{`⟨*code*⟩`}` defines the `\_nokvx:`⟨*dict*⟩`:`⟨*key*⟩ macro.
The `\trykv{`⟨*key*⟩`}{`⟨*code*⟩`}` returns unexpanded value of ⟨*key*⟩ if declared, else it runs ⟨*code*⟩.
The `\kv{`⟨*key*⟩`}` returns expanded value of ⟨*key*⟩ if declared, else returns expanded value of `.notdef` key else expands `\_kvunknown`.
The `\iskv{`⟨*key*⟩`}\iftrue` (or `\iffalse`) is the test, if the ⟨*key*⟩ is defined in current ⟨*dict*⟩.

keyval.opm
```
\_def\_readkv#1{\_ea\_def\_ea\_tmpb\_ea{#1,}%
   \_replstring\_tmpb{= }{=}\_replstring\_tmpb{ =}{=}\_replstring\_tmpb{ ,}{,}%
   \_ea \_nospaceafter \_ea\_kvscan\_tmpb\_fin}
\_def\_kvscan#1,#2{\_ifx^#1^\_else \_kvsd #1==\_fin \_fi
   \_ifx\_fin#2\_empty \_ea\_ignoreit \_else\_ea\_useit \_fi {\_kvscan#2}}
\_def\_kvsd#1=#2=#3\_fin{\_sdef{\_kvcs#1}{#2}%
   \_trycs{_kvx:\_the\_kvdict:#1}%
          {\_trycs{_nokvx:\_the\_kvdict}{\_ea\_ignoreit}{#1}\_ea\_ignoreit}{#2}}
\_def\_kvx#1#2{\_sdef{_kvx:\_the\_kvdict:#1}##1{#2}}
\_def\_nokvx#1{\_sdef{_nokvx:\_the\_kvdict}##1\_ea\_ignoreit##2{#1}}
\_def\_trykv#1{\_ea\_trykvA \_begincsname\_kvcs#1\_endcsname \_ignoreit}
\_def\_trykvA#1{\_ifx #1\_ignoreit \_ea\_useit\_else \_unexpanded\_ea{#1}\_fi}
\_def\_kv#1{\_expanded{\_trykv{#1}{\_trykv{.\_csstring\notdef}{\_kvunknown}}}}
\_def\_iskv#1#2{#2\_else\_ea\_unless\_fi \_ifcsname\_kvcs#1\_endcsname}
\_def\_kvcs{_kv:\_the\_kvdict:}
\_def\_kvunknown{???}

\public \readkv \kvx \nokvx \kv \trykv \iskv ;
```

2.10 Plain TEX macros

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

plain-macros.opm
```
\_codedecl \magstep {Macros from plain TeX <2022-10-11>} % preloaded in format
```

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

```
\addto \dospecials{\do\_}
```

`\active` is character constant 13, we can use it in the context `\catcode`⟨*character*⟩`=\active`.

plain-macros.opm
```
\_def\_dospecials {\do\do\do\do\do\do%
  \do\do\do^K\do^A\do\do}
\_chardef\_active = 13

\_public \dospecials \active ;
```

The shortcuts `\chardef\@one` is not defined in OpTeX. Use normal numbers instead of such obscurities. The `\magstep` and `\magstephalf` are defined with `\space`, (no `\relax`), in order to be expandable.

plain-macros.opm

```
\_def \_magstephalf{1095 }
\_def \_magstep#1{\_ifcase#1 1000\_or 1200\_or 1440\_or 1728\_or 2074\_or 2488\_fi\_space}
\_public \magstephalf \magstep ;
```

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

plain-macros.opm

```
\_def^M{} % control <return> = control <space>
\_def^I{} % same for <tab>

\_def\lq{`} \def\rq{'}
\_def\lbrack{[} \_def\rbrack{]} % They are only public versions.
% \catcode`^L=\active \outer\def^^L{\par} % ascii form-feed is "\outer\par" % obsolete

\_let\_endgraf=\_par \_let\_endline=\_cr
\_public \endgraf \endline ;
```

Plain TeX classical `\obeylines` and `\obeyspaces`.

plain-macros.opm

```
% In \obeylines, we say `\let^^M=\_par' instead of `\def^^M{\_par}'
% since this allows, for example, `\let\_par=\cr \obeylines \halign{...'
{\_catcode`^M=13 % these lines must end with %
  \_gdef\_obeylines{\_catcode`^M=13\_let^^M\_par}%
  \_glet^^M=\_par} % this is in case ^^M appears in a \write
\_def\_obeyspaces{\_catcode`=13 }
{\_obeyspaces\_glet =\_space}
\_public \obeylines \obeyspaces ;
```

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

plain-macros.opm

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

\_public \thinspace \negthinspace \enspace \enskip \quad \qquad \smallskip
   \medskip \bigskip \nointerlineskip \offinterlineskip ;

\_def\_topglue {\_nointerlineskip\_vglue-\_topskip\_vglue} % for top of page
\_def\_vglue {\_afterassignment\_vglA \_skip0=}
\_def\_vglA {\_par \_dimen0=\_prevdepth \_hrule height0pt
  \_nobreak\_vskip\_skip0 \_prevdepth=\_dimen0 }
\_def\_hglue {\_afterassignment\_hglA \_skip0=}
\_def\_hglA {\_leavevmode \_count255=\_spacefactor \_vrule width0pt
  \_nobreak\_hskip\_skip0 \_spacefactor=\_count255 }
\_protected\_def~{\_penalty10000 } % tie
\_protected\_def\_slash {/\_penalty\_exhyphenpenalty} % a `/' that acts like a `-'

\_public \topglue \vglue \hglue \slash ;
```

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

plain-macros.opm

```
\_protected\_def \_break {\_penalty-10000 }
\_protected\_def \_nobreak {\_penalty10000 }
\_protected\_def \_allowbreak {\_penalty0 }
\_protected\_def \_filbreak {\_par\_vfil\_penalty-200\_vfilneg}
\_protected\_def \_goodbreak {\_par\_penalty-500 }
```

```
\_protected\_def \_eject {\_par\_break}
\_protected\_def \_supereject {\_par\_penalty-20000 }
\_protected\_def \_dosupereject {\_ifnum \_insertpenalties>0 % something is being held over
  \_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`.

plain-macros.opm

```
\_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. The `\strut` macro puts the `\strutbox`.

plain-macros.opm

```
\_newbox\_strutbox
\_setbox\_strutbox=\_hbox{\_vrule height8.5pt depth3.5pt width0pt}
\_def \_strut {\_relax\_ifmmode\_copy\_strutbox\_else\_unhcopy\_strutbox\_fi}

\_public \strutbox \strut ;
```

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

plain-macros.opm

```
\_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 similar macros are defined here: `\hang`, `\textindent`, `\item`, `\itemitem`, `\narrower`, `\raggedright`, `\ttraggedright`, `\leavevmode`.

plain-macros.opm

```
\_def \_hang {\_hangindent\_parindent}
\_def \_textindent #1{\_indent\_llap{#1\_enspace}\_ignorespaces}
\_def \_item {\_par\_hang\_textindent}
\_def \_itemitem {\_par\_indent \_hangindent2\_parindent \_textindent}
\_def \_narrower {\_advance\_leftskip\_parindent
   \_advance\_rightskip\_parindent}
\_def \_raggedright {\_rightskip=0pt plus2em
   \_spaceskip=.3333em \_xspaceskip=.5em\_relax}
\_def \_ttraggedright {\_tt \_rightskip=0pt plus2em\_relax} % for use with \tt only
\_def \_leavevmode {\_unhbox\_voidbox} % begins a paragraph, if necessary

\_public \hang \textindent \item \itemitem \narrower \raggedright \ttraggedright \leavevmode ;
```

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

plain-macros.opm

```
%\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
\_protected\_def\_{\_relax \_ifmmode \_hbox{_}\_else _\_fi}
\_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 when Unicode font family is loaded, Unicodes are used instead old plain TeX settings.

plain-macros.opm

```
\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 `\ooalign`, `\d`, `\b`, `\c`, `\dots`, are defined for backward compatibility.

plain-macros.opm

```
\_def \_oalign #1{\_leavevmode\_vtop{\_baselineskip=\_zo \_lineskip=.25ex
   \_ialign{##\_crcr#1\_crcr}}}
\_def \_oalignA {\_lineskiplimit=\_zo \_oalign}
\_def \_ooalign {\_lineskiplimit=-\_maxdimen \_oalign} % chars over each other
\_def \_shiftx #1{\_dimen0=#1\_kern\_ea\_ignorept \_the\_fontdimen1\_font
   \_dimen0 } % kern by #1 times the current slant
\_def \_d #1{{\_oalignA{\_relax#1\_crcr\_hidewidth\_shiftx{-1ex}.\_hidewidth}}}
\_def \_b #1{{\_oalignA{\_relax#1\_crcr\_hidewidth\_shiftx{-3ex}%
   \_vbox to.2ex{\_hbox{\_char\_macron}\_vss}\_hidewidth}}}
\_def \_c #1{{\_setbox0=\_hbox{#1}\_ifdim\_ht0=1ex\_accent\_cedilla #1%
   \_else\_ooalign{\_unhbox0\_crcr\_hidewidth\_cedilla\_hidewidth}\_fi}}
\_def\_dots{\_relax\_ifmmode\_ldots\_else$\_math\_ldots\_thinsk$\_fi}
\_public \oalign \ooalign \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.

The `\_uniaccents` macro redeclares codes for accents and it is run when Unicode font family is loaded.

plain-macros.opm

```
\_def \_oldaccents {%
```

```
   \_def##1{{\_accent\_tgrave  ##1}}%
   \_def##1{{\_accent\_tacute  ##1}}%
   \_def\v##1{{\_accent\_caron  ##1}}%
   \_def\u##1{{\_accent\_tbreve  ##1}}%
   \_def##1{{\_accent\_macron ##1}}%
   \_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="00B8
   \_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$}\_hfill
  \_mkern-7mu\_smash-$}

\_mathchardef \_braceld="37A \_mathchardef \_bracerd="37B
\_mathchardef \_bracelu="37C \_mathchardef \_braceru="37D
\_def \_downbracefill {$\_math \_setbox0=\_hbox{$\_braceld$}%
  \_braceld \_leaders\_vrule height\_ht0 depth\_zo \_hfill \_braceru
  \_bracelu \_leaders\_vrule height\_ht0 depth\_zo \_hfill \_bracerd$}
\_def \_upbracefill {$\_math \_setbox0=\_hbox{$\_braceld$}%
  \_bracelu \_leaders\_vrule height\_ht0 depth\_zo \_hfill \_bracerd
  \_braceld \_leaders\_vrule height\_ht0 depth\_zo \_hfill \_braceru$}

\_public \hrulefill \dotfill
   \rightarrowfill \leftarrowfill \downbracefill \upbracefill ;
```

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

plain-macros.opm

```
\_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 when `\language=0`. We do the same.

plain-macros.opm

```
\_lefthyphenmin=2 \_righthyphenmin=3 % disallow x- or -xx breaks
\_input hyphen  % en(USenglish) patterns from TeX82
```

2.11 Preloaded fonts for text mode

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

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

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

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

fonts-preload.opm

```
\_codedecl \tenrm {Latin Modern fonts (EC) preloaded <2022-02-12>} % preloaded in format

\_ifx\fontspreload\_relax
   \_let\_tenrm=\_nullfont \_let\_tenbf=\_nullfont \_let\_tenit=\_nullfont
   \_let\_tenbi=\_nullfont \_let\_tentt=\_nullfont
\_else
   % Only few text fonts are preloaded:
   % allow missing fonts during format generation
   \_suppressfontnotfounderror=1
   \_font\_tenrm=ec-lmr10   % roman text
   \_font\_tenbf=ec-lmbx10  % boldface extended
   \_font\_tenit=ec-lmri10  % text italic
   \_font\_tenbi=ec-lmbxi10 % bold italic
   \_font\_tentt=ec-lmtt10  % typewriter
   \_suppressfontnotfounderror=0
\_fi

\_tenrm

\_public \tenrm \tenbf \tenit \tenbi \tentt ;
```

2.12 Using `\font` primitive directly

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

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

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

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

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

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

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

`\initunifonts` loads the implementation of the `\font` primitive from `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 `tfm` 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` of `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 `tfm` fonts. We strongly recommend to prefer OpenType format over Type1 format over `tfm` format. The last one doesn't support Unicode. If there is nothing else left and you must to use `tfm`, then you must to implement re-encoding from Unicode to the `tfm` encoding at macro level, see the OpTEX 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; OpTEX 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:

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

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

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

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

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

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

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

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

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

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

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

2.12.2 The `\font`-like commands summary

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

2.12.3 The `\fontlet` declarator

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

```
\fontlet \⟨new font switch⟩ = \⟨given font switch⟩ ⟨size spec⟩
example:
\fontlet \bigfont = \_tenbf at15pt
```

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

```
\fontlet \⟨new font switch⟩ = \font ⟨size spec⟩
```

2.12.4 Optical sizes

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

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

See also section 2.13.12.

2.12.5 Font rendering

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

The OpTEX format uses `luahbtex` engine by default (from 2025). It means that 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.

fonts-resize.opm

```
\_codedecl \setfontsize {Font resizing macros <2022-11-08>} % preloaded in format
```

`\initunifonts` macro extends LuaTEX's font capabilities, in order to be able to load Unicode fonts. Unfortunately, this part of OpTEX depends on the `luaotfload` package, which adapts ConTEXt's generic font loader for 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`.

fonts-resize.opm

```
\_protected\_def \_initunifonts {%
   \_directlua{%
      require('luaotfload-main')
      luaotfload.main()
      optex.hook_into_luaotfload()
   }%
   \_glet \_fmodtt=\_unifmodtt  % use \_ttunifont for \tt
   \_glet \_initunifonts=\_relax % we need not to do this work twice
   \_glet \initunifonts=\_relax
}
\_protected\_def \_ufont {\_initunifonts \_font}

\_public \initunifonts \ufont ;
```

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.

fonts-resize.opm

```
\_newdimen \_optsize         \_optsize=10pt
\_newdimen \_defaultoptsize  \_defaultoptsize=10pt
\_newdimen\_lastmagsize

\_def\_setfontsize #1{%
   \_edef\_sizespec{#1}%
   \_ea \_setoptsize \_sizespec\_relax
}
\_def\_setoptsize {\_isnextchar a{\_setoptsizeA}
                                 {\_isnextchar m{\_setoptsizeC}{\_setoptsizeB}}}
```

```
\_def\_setoptsizeA at#1\_relax{\_optsize=#1\_relax\_lastmagsize=\_optsize}   % at<dimen>
\_def\_setoptsizeB scaled#1\_relax{\_optsize=\_defaultoptsize\_relax} % scaled<scalenum>
\_def\_setoptsizeC mag#1\_relax{%
   \_ifdim\_lastmagsize>\_zo \_optsize=\_lastmagsize \_else \_optsize=\_pdffontsize\_font \_fi
   \_optsize=#1\_optsize
   \_lastmagsize=\_optsize
   \_edef\_sizespec{at\_the\_optsize}%
}
\_public \setfontsize \defaultoptsize ;
```

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

fonts-resize.opm

```
\_edef\_stringat{\_string a\_string t}
\_edef\_xfontname#1{\_unexpanded{\_ea\_xfontnameA\_fontname}#1 \_stringat\_relax}
\_expanded{\_def\_noexpand\_xfontnameA#1 \_stringat#2\_relax}{#1}
```

`\fontlet` ⟨*font switch A*⟩ ⟨*font switch B*⟩ ⟨*size spec*⟩ does

`\font` ⟨*font switch A*⟩ `= {`⟨*font file name*⟩`}` ⟨*size spec*⟩

Note, that the `\_xfontname` output is converted due to optical size data using `\_optfn`.

fonts-resize.opm

```
\_protected\_def \_fontlet #1#2{\_ifx #2=\_ea\_fontlet \_ea#1\_else
   \_ea\_font \_ea#1\_expanded{{\_optfn{\_xfontname#2}}}\_fi}
\_public \xfontname \fontlet ;
```

`\newcurrfontsize` ⟨*size spec*⟩ does `\fontlet` ⟨*saved switch*⟩`=\font` ⟨*size spec*⟩`\_relax` ⟨*saved switch*⟩. It changes the current font at the given ⟨*size spec*⟩.
`\resizethefont` is implemented by `\newcurrfontsize` using data from the `\_sizespec` macro.
`\sfont` has the same syntax like `\font` primitive, but declares a macro which selects the font and sets its size properly dependent on the current size.

fonts-resize.opm

```
% \newcurrfontsize{at25pt}
\_def \_newcurrfontsize {\_ea\_newcurrfontsizeA \_csname \_ea\_csstring \_the\_font \_endcsname}
\_def \_newcurrfontsizeA #1#2{\_fontlet #1\_font #2\_relax \_fontloaded#1#1}
\_protected\_def \_resizethefont {\_newcurrfontsize\_sizespec}
\_protected\_def \_sfont #1{%
   \_protected\_edef #1{\_csname _sfont:\_csstring#1\_endcsname \_resizethefont}%
   \_initunifonts \_ea\_font \_csname _sfont:\_csstring#1\_endcsname
}
\_public \newcurrfontsize \resizethefont \sfont ;
```

The `\_regtfm` ⟨*font id*⟩ ⟨*optical size data*⟩ registers optical sizes data directly by the font file names. This can be used for `tfm` files or OpenType files without various font features. See also `\_regoptsizes` in section 2.13.12. The `\_regtfm` command saves the ⟨*optical size data*⟩ concerned to the ⟨*font id*⟩. The ⟨*optical size data*⟩ is in the form as shown below in the code where `\_regtfm` is used.
The `\_optfn` ⟨*fontname*⟩ expands to the ⟨*fontname*⟩ or to the corrected ⟨*fontname*⟩ read from the ⟨*optical size data*⟩ registered by `\_regtfm`. It is used in the `\fontlet` macro.

The implementation detail: The `\_reg:`⟨*font id*⟩ is defined as the ⟨*optical size data*⟩ and all control sequences `\_reg:`⟨*fontname*⟩ from this data line have the same meaning because of the `\_reversetfm` macro. The `\_optfn` expands this data line and apply `\_runoptfn`. This macro selects the right result from the data line by testing with the current `\_optsize` value.

fonts-resize.opm

```
\_def\_regtfm #1 0 #2 *{\_ea\_def \_csname _reg:#1\_endcsname{#2 16380 \_relax}%
  \_def\_tmpa{#1}\_reversetfm #2 * %
}
\_def\_reversetfm #1 #2 {% we need this data for \_setmathfamily
   \_ea\_let\_csname _reg:#1\_ea\_endcsname
   \_csname _reg:\_tmpa\_endcsname
   \_if*#2\_else \_ea\_reversetfm \_fi
}
\_def\_optfn #1{%
   \_ifcsname _reg:#1\_endcsname
      \_ea\_ea\_ea \_runoptfn
      \_csname _reg:#1\_ea\_endcsname
   \_else
      #1%
   \_fi
}
\_def\_runoptfn #1 #2 {%
   \_ifdim\_optsize<#2pt #1\_ea\_ignoretfm\_else \_ea\_runoptfn
\_fi
}
\_def\_ignoretfm #1\_relax{}
```

Optical sizes data for preloaded 8bit Latin Modern fonts:

`fonts-resize.opm`

```
\_regtfm lmr  0 ec-lmr5 5.5 ec-lmr6 6.5 ec-lmr7 7.5 ec-lmr8 8.5 ec-lmr9 9.5
                         ec-lmr10 11.1 ec-lmr12 15 ec-lmr17 *
\_regtfm lmbx 0 ec-lmbx5 5.5 ec-lmbx6 6.5 ec-lmbx7 7.5 ec-lmbx8 8.5 ec-lmbx9 9.5
                         ec-lmbx10 11.1 ec-lmbx12 *
\_regtfm lmri 0 ec-lmri7 7.5 ec-lmri8 8.5 ec-lmri9 9.5 ec-lmri10 11.1 ec-lmri12 *
\_regtfm lmtt 0 ec-lmtt8 8.5 ec-lmtt9 9.5 ec-lmtt10 11.1 ec-lmtt12 *
```

2.13 The Font Selection System

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

2.13.1 Terminology

We distinguish between

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

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

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

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

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

2.13.2 Font families, selecting fonts

The `\fontfam [`⟨*Font Family*⟩`]` opens the relevant font family file where the ⟨*Font Family*⟩ is declared. The family selector is defined here by rules described in the section 2.13.11. Font modifiers and variant selectors may be declared here. The loaded family is set as current and `\rm` variant selector is processed.

When `\fontfam [`⟨*Font Family*⟩`]` is used and the given family isn't found in the current TeX system and the ⟨*Font Family*⟩ is previously declared by `\fontfamsub[`⟨*Font Family*⟩`] [`⟨*Other Family*⟩`]` then OpTeX does the given substitution and runs `\fontfam[`⟨*Other Family*⟩`]`.

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

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

You can open more font families by more `\fontfam` commands. Then the general method to selecting the individual font is:

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

For example:

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

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

You can use the `\setfontsize` `{`⟨*size spec*⟩`}` command in the same sense as other font modifiers. It saves information about font size to the font context. See section 2.12.1. Example:

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

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

2.13.3 Math Fonts

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

If you want to use your specially selected Unicode-math font then use `\loadmath` `{[`⟨*font file*⟩`]}` or `\loadmath` `{`⟨*font name*⟩`}` before first `\fontfam` is used.

2.13.4 Declaring font commands

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

- Font switches can be decared by `\font` primitive (see section 2.12) or by `\fontlet` command (see section 2.12.3) or by `\fontdef` command (see sections 2.13.5). When the font switches are used then they select the given font independently of the current font context. They can be used in `\output` routine (for example) because we need to set fixed fonts in headers and footers.
- Variant selectors are `\rm`, `\bf`, `\it`, `\bi`, `\tt` and `\currvar`. More variant selectors can be declared by `\famvardef` command. They select a font dependent on the current font context, see section 2.13.6. The `\tt` selector is documented in section 2.13.7.
- Font modifiers are "built-in" or declared by `\moddef` command. They do modifications in the font context but don't select any font.
  - "built-in" font modifiers are `\setfontsize` (see section 2.12.1), `\setff` (see section 2.13.9), `\setletterspace` and `\setwordspace` (see section 2.13.10). They are independent of font family.
  - Font modifiers declared by `\moddef` depend on the font family and they are typically declared in font family files, see section 2.13.11.

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

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

2.13.5 The `\fontdef` declarator in detail

You can declare \⟨*font-switch*⟩ by the `\fontdef` command.

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

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

The resulting \⟨*font-switch*⟩ declared by `\fontdef` is "fixed font switch" independent of the font context. More exactly, it is a fixed font switch when it is *used*. But it can depend on the current font modifiers and font family and given font modifiers when it is *declared*.

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

2.13.6 The `\famvardef` declarator

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

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

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

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

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

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

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

A \⟨*family selector*⟩ can be written before ⟨*font modifiers*⟩ in the `\famvardef` parameter. Then the \⟨*new variant selector*⟩ is declared in the current family but it can use fonts from another family represented by the \⟨*family selector*⟩.

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

```
\fontfam[Heros]  \fontfam[Termes]

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

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

The variant selectors (declared by `\famvardef`) or font modifiers (declared by `\moddef`) are (typically) control sequences in the public namespace (`\mf`, `\caps`). They are most often declared in font family files and they are loaded by `\fontfam`. A conflict with such names in the public namespace can be here. For example: if `\mf` is defined by a user and then `\fontfam[Roboto]` is used then `\famvardef\mf` is performed for Roboto family and the original meaning of `\mf` is lost. But OpTEX prints warning about it. There are two cases:

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

2.13.7 The `\tt` variant selector

`\tt` is an additional special variant selector which is defined as "select typewriter font independently of the current font family". By default, the typewriter font-face from LatinModern font family is used. When using this default `\tt` selector you need not to load the LatinModern family (by `\fontfam[lm]`) because the font file is explicitly specified in the macro `\_ttunifont`. The default `\tt` selector scales this font without respecting optical sizes.

The `\tt` variant selector is used in OpTEX internal macros `\_ttfont` (verbatim texts), `\_urlfont` (printing URL's), `\_printii` (printing index entry when doc macros are activated) and `\ttraggedright` (plain TEX macro).

The `\tt` variant selector can be re-defined by `\famvardef`. For example:

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

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

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

The `\famvardef\tt` behaves differently than `\famvardef\anythingelse`: it creates the `\tt` variant selector independent of current font family and it redefines both macros `\tt` and `\_tt` in order to all internal OpTEX macros based on this selector are working with the new selected font.

If you load `\fontfam[lm]` and declare `\famvardef\tt{\LMfonts\ttset\setff{-tlig}\rm}` then you get almost the same behavior as the default `\tt` selector does with one difference: a new `\tt` selector respects optical sizes declared in the LatinModern font family.

You can implement ex-height correction of `\tt` font with respect to your font family used for general text. For example:

```
\load[lm]
\famvardef\tt{\LMfonts\ttset\setfontsize{mag1.1}\setff{-tlig}\rm}
... or ...
\let\oritt=\tt
\famvardef\tt{\setfontsize{mag1.1}\oritt}
```

The second alternative uses the default font from `\_ttunifont` without respecting optical sizes.

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

2.13.8 Font commands defined by `\def`

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

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

```
\def\titlefont{\setfontsize{at14pt}\bf \let\it\bi}
...
{\titlefont Title in bold 14pt font and {\it bold 14pt italics} too}
```

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

2.13.9 Modifying font features

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

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

```
\fontfam[LinLibertine]
\famvardef \fraclig {\setff{+frac}\currvar}
Compare 1/2 or 1/10 \fraclig to 1/2 or 1/10.
```

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

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

2.13.10 Special font modifiers

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

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

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

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

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

You can use `\setff` with other font features provided by LuaTEX and `luaotfload` package (see documentation of `loaotfload` package for more information):

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

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

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

2.13.11 How to create the font family file

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

f-heros.opm

```
\_famdecl [Heros] \Heros {TeX Gyre Heros fonts based on Helvetica}
     {\caps \cond} {\rm \bf \it \bi} {FiraMath}
     {[texgyreheros-regular]}
     {\_def\_fontnamegen{[texgyreheros\_condV-\_currV]:\_capsV\_fontfeatures}}

\_wlog{\_detokenize{%
Modifiers:^^J
 \caps ...... caps & small caps^^J
 \cond ...... condensed variants^^J
}}

\_moddef \resetmod {\_fsetV caps={},cond={} \_fvars regular bold italic bolditalic }
\_moddef \caps     {\_fsetV caps=+smcp;\_ffonum; }
\_moddef \nocaps   {\_fsetV caps={} }
\_moddef \cond     {\_fsetV cond=cn }
\_moddef \nocond   {\_fsetV cond={} }

\_initfontfamily % new font family must be initialized

\_ifmathloading
   \_loadmath {[FiraMath-Regular]}
   \_addUmathfont \_xits {[XITSMath-Regular]}{} {[XITSMath-Bold]}{} {}
   \_addto\_frak{\_fam\_xits}\_addto\_cal{\_fam\_xits} \_public \frak \cal ;
   % \bf, \bi from FiraMath:
   \_let\_bsansvariables=\_bfvariables
   \_let\_bsansGreek=\_bfGreek
   \_let\_bsansgreek=\_bfgreek
   \_let\_bsansdigits=\_bfdigits
   \_let\_bisansvariables=\_bivariables
   \_let\_bisansgreek=\_bigreek
   % \_resetmathchars <fam-number> <list of \Umathchardef csnames> ;
   \_mathchars \_xits {\bigtriangleup \bigblacktriangleup \blacktriangle
       \vartriangle \smallblacktriangleright
       \unicodevdots \unicodeadots \unicodeddots} % ... etc. you can add more
\_fi
```

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

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

  _famdecl [⟨*Name of family*⟩] \⟨*Familyselector*⟩ {⟨*comments*⟩}
  {⟨*modifiers*⟩} {⟨*variant selectors*⟩} {⟨*comments about math fonts*⟩}
  {⟨*font-for-testing*⟩}
  {_def_fontnamegen{⟨*font name or font file name generated*⟩}}

  This writes information about font family at the terminal and prevents loading such file twice. Moreover, it probes existence of ⟨*font-for-testing*⟩ in your system. If it doesn't exist, the file loading is skipped with a warning on the terminal. The `\_ifexistfam` macro returns false in this case. The `\_fontnamegen` macro must be defined in the last parameter of the `\_famdecl`. More about it is documented below.
- You can use `\_wlog{\_detokenize{...` to write additional information into a log file.
- You can declare optical sizes using `\_regoptsizes` if there are more font files with different optical sizes (like in Latin Modern). See `f-lmfonts.opm` file for more information about this special feature.
- Declare font modifiers using `\moddef` if they are present. The `\resetmod` must be declared in each font family.
- Check if all your declared modifiers do not produce any space in horizontal mode. For example check: `X\caps Y`, the letters `XY` must be printed without any space.
- Optionally, declare new variants by the `\famvardef` macro.
- Run `\_initfontfamily` to start the family (it is mandatory).
- If math font should be loaded, use _loadmath{⟨*math font*⟩}.

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

```
\font \⟨font-switch⟩ {\_fontnamegen} \_sizespec
```

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

Example 1

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

```
\_moddef\resetmod{\_fvars Regular Bold Italic BoldItalic }
```

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

```
\_famdecl ...
    {\def\_fontnamegen{[xx-\_currV]}}
```

The following auxiliary macros are used here:

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

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

See more advanced examples are in `f-`⟨*family*⟩`.opm` files.

Example 2

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

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

```
\_fsetV ⟨word-a⟩=⟨value-a⟩,⟨word-b⟩=⟨value-b⟩  ...etc. terminated by a space
```

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

Example 3

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

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

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

The `\_onlyif` macro

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

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

The `\moddef` macro

has the syntax `\moddef\<modifier>{`⟨*what to do*⟩`}`. It does more things than simple `\_def`:

- The modifier macros are defined as `\_protected`.
- The modifier macros are defined as family-dependent.
- If the declared control sequence is defined already (and it is not a font modifier) then it is re-defined with a warning.

The `\famvardef` macro has the same features.

The \⟨*Familyselector*⟩ is defined by the `\_famdecl` macro as:

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

The `\_initfontfamily`
must be run after modifier's decaration. It runs the \⟨*Familyselector*⟩ and it runs `\_rm`, so the first font from the new family is loaded and it is ready to use it.

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

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

The name of \⟨*Familyselector*⟩ should begin with an uppercase letter.

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

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

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

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

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

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

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

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

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

2.13.12 How to write the font family file with optical sizes

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

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

```
\_regoptsizes lmr.r 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` ⟨*internal-template*⟩ ⟨*general-ouput-template*⟩ ⟨*resizing-data*⟩

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 ⟨*resizing-data*⟩ virtually begins with zero `<0`, but it is not explicitly written. The right part of ⟨*resizing-data*⟩ must be terminated by `<*` which means ”less than infinity”.

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

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

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

The `\_optname` macro can be used even if no 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:

f-poltawski.opm

```
\_famdecl [Poltawski] \Poltawski {Antykwa Poltawskiego, Polish traditional font family}
      {\light \noexpd \expd \eexpd \cond \ccond \osize \caps} {\rm \bf \it \bi} {}
      {[antpolt-regular]}
      {\_def\_fontnamegen {[antpolt\_liV\_condV-\_currV]\_capsV\_fontfeatures}}

\_wlog{\_detokenize{%
Modifiers:^^J
 \light ..... light weight, \bf,\bi=semibold^^J
 \noexpd .... no expanded, no condensed, designed for 10pt size (default)^^J
 \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-switches 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 \eexpd    {\_fsetV cond=expd }
\_moddef \expd     {\_fsetV cond=semiexpd }
\_moddef \cond     {\_fsetV cond=semicond }
\_moddef \ccond    {\_fsetV cond=cond }
```

```
\_moddef \caps     {\_fsetV caps=+smcp;\_ffonum; }
\_moddef \nocaps   {\_fsetV caps={} }
\_moddef \osize    {\_def\_fontnamegen{[antpolt\_liV\_optname{x}-\_currV]:\_capsV\_fontfeatures}%
                    \_regoptsizes x ? expd <7 semiexpd <9 {} <11.1 semicond <15 cond <*}

\_initfontfamily % new font family must be initialized
```

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

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

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

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

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

`fams-ini.opm`

```
% Version <2022-10-18>. Loaded in format and secondly on demand by \fontfam[catalog]

\_famtext {Special name for printing a catalog :}

\_faminfo [Catalogue] {Catalogue of all registered font families} {fonts-catalog} {}
\_famalias [Catalog]

\_famsrc {CTAN}
\_famtext {Computer Modern like family:}

\_famfrom {GUST}
\_faminfo [Latin Modern] {TeX Gyre fonts based on Computer Modern} {f-lmfonts}
   { -,\nbold,\sans,\sans\nbold,\slant,\ttset,\ttset\slant,\ttset\caps,%
       \ttprop,\ttprop\bolder,\quotset: {\rm\bf\it\bi}
       \caps: {\rm\it}
       \ttlight,\ttcond,\dunhill: {\rm\it} \upital: {\rm} }
\_famalias [LMfonts] \_famalias [Latin Modern Fonts] \_famalias [lm]

\_famtext {TeX Gyre fonts based on Adobe 35:}

\_faminfo [Termes] {TeX Gyre Termes fonts based on Times} {f-termes}
   { -,\caps: {\rm\bf\it\bi} }
\_famalias [Times]

\_faminfo [Heros] {TeX Gyre Heros fonts based on Helvetica} {f-heros}
   { -,\caps,\cond,\caps\cond: {\rm\bf\it\bi} }
```

... etc.

The `\_faminfo` command has the syntax:

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

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

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

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

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

2.13.14 Implementation of the Font Selection System

fonts-select.opm

```
\_codedecl \fontfam {Fonts selection system <2023-06-16>} % preloaded in format
```

The main principle of the Font Selection System is: run one or more modifiers followed by `\fontsel`. Modifiers save data and `\fontsel` selects the font considering saved data. Each basic variant selector `\rm`, `\bf`, `\it`, `\bi`, and `\tt` runs internal variant modifier `\_fmodrm`, `\_fmodbf`, `\_fmodit`, `\_fmodbi` and `\_fmodtt`. These modifiers save their data to the `\_famv` macro which is `rm` or `bf` or `it` or `bi` or `tt`. The `\currvar` selector is `\fontsel` by default, but variant selectors declared by `\famvardef` change it.

fonts-select.opm

```
\_def\_famv{rm}  % default value
\_protected\_def \_fmodrm {\_def\_famv{rm}}
\_protected\_def \_fmodbf {\_def\_famv{bf}}
\_protected\_def \_fmodit {\_def\_famv{it}}
\_protected\_def \_fmodbi {\_def\_famv{bi}}
\_protected\_def \_fmodtt {\_def\_famv{tt}}

\_protected\_def \_rm {\_fmodrm \_fontsel \_marm}
\_protected\_def \_bf {\_fmodbf \_fontsel \_mabf}
\_protected\_def \_it {\_fmodit \_fontsel \_mait}
\_protected\_def \_bi {\_fmodbi \_fontsel \_mabi}
\_protected\_def \_tt {\_fmodtt \_fontsel \_matt}
\_protected\_def \_currvar {\_fontsel} \_protected\_def \currvar{\_currvar}
\_public \rm \bf \it \bi \tt ;
```

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

a) `\letfont` ⟨*font switch*⟩ `= \savedswitch \_sizespec`
b) `\font` ⟨*font switch*⟩ `= \fontnamegen \_sizespec`

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

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

fonts-select.opm

```
\_protected\_def \_fontsel {%
   \_ifx\_fontnamegen\_undefined % \fontfam was not used
      \_ea\_let \_ea\_tmpf \_csname _ten\_famv\_endcsname
      \_ea\_fontlet \_csname _ten\_xfamv\_endcsname \_tmpf \_sizespec
   \_else % \fontfam is used
      \_ea\_font \_csname _ten\_xfamv\_endcsname {\_fontnamegen}\_sizespec
   \_fi \_relax
   \_ea \_fontselA \_csname _ten\_xfamv\_endcsname
}
\_def\_fontselA #1{%
   \_protected\_def \_currvar {\_fontsel}% default value of \_currvar
   \_logfont #1%   font selecting should be logged.
   \_setwsp  #1%    wordspace setting
   \_fontloaded #1% initial settings if font is loaded firstly
   #1% select the font
}
\_def \_logfont #1{}
\_def \_xfamv {\_famv}

\_public \fontsel ;
```

If a font is loaded by macros `\fontsel` or `\resizethefont` then the `\_fontloaded`⟨*font switch*⟩ is called immediately after it. If the font is loaded first then its `\skewchar` is equal to -1. We run

`\_newfontloaded`⟨*font switch*⟩ and set `\skewchar=-2` in this case. A user can define a `\_newfontloaded` macro. We are sure that `\_newfontloaded` macro is called only once for each instance of the font given by its name, OTF features and size specification. The `\skewchar` value is globally saved to the font (like `\fontdimen`). If it is used in math typesetting then it is set to a positive value.
The `\_newfontloaded` should be defined for micro-typographic configuration of fonts, for example. The `mte.opm` package uses it. See also OpTeX trick 0058.

fonts-select.opm

```
\_def\_fontloaded #1{\_ifnum\_skewchar#1=-1 \_skewchar#1=-2 \_newfontloaded#1\_fi}
\_def\_newfontloaded #1{}
```

`\_ttunifont` is default font for `\tt` variant when `\initunifonts` is declared. User can re-define it or use `\famvardef\tt`. The `\_unifmodtt` macro is used instead `\_fmodtt` after `\initunifonts`. It ignores the loading part of the following `\fontsel` and do loading itself.

fonts-select.opm

```
\_def\_ttunifont{[lmmono10-regular]:\_fontfeatures-tlig;}
\_def\_unifmodtt\_fontsel{%  ignore following \_fontsel
   \_ea\_font \_csname _ten\_ttfamv\_endcsname {\_ttunifont}\_sizespec \_relax
   \_ea\_fontselA \_csname _ten\_ttfamv\_endcsname
   \_def \_currvar{\_tt}%
}
\_def\_ttfamv{tt}
```

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

fonts-select.opm

```
\_def \_tryloadrm\_tenrm {\_fmodrm \_fontsel}
\_def \_tryloadbf\_tenbf {\_fmodbf \_fontsel}
\_def \_tryloadit\_tenit {\_fmodit \_fontsel}
\_def \_tryloadbi\_tenbi {\_fmodbi \_fontsel}
\_def \_tryloadtt\_tentt {\_fmodtt \_fontsel}
\_def \_reloading {}
```

The `\_famdecl` `[`⟨*Family Name*⟩`]` `\`⟨*Famselector*⟩ `{`⟨*comment*⟩`}` `{`⟨*modifiers*⟩`}` `{`⟨*variants*⟩`}` `{`⟨*math*⟩`}` `{`⟨*font for testing*⟩`}` `{\def\_fontnamegen{`⟨*data*⟩`}}` runs `\initunifonts`, then checks if `\`⟨*Famselector*⟩ is defined. If it is true, then closes the file by `\endinput`. Else it defines `\`⟨*Famselector*⟩ and saves it to the internal `\_f:`⟨*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 `\_fontnamegen` macro. The `\_mathfaminfo` is saved for usage in the catalog.

fonts-select.opm

```
\_def\_famdecl [#1]#2#3#4#5#6#7#8{%
   \_initunifonts \_unichars \_uniaccents
   \_unless\_ifcsname _f:\_csstring#2:main.fam\_endcsname
      \_isfont{#7}\_iffalse
         \_opwarning{Family [#1] skipped, font "#7" not found}\_endinput
         \_ifcsname _fams:\_famfile\_endcsname \_famsubstitute \_fi
      \_else
         \_edef\_currfamily {\_csstring #2}\_def\_mathfaminfo{#6}%
         \_wterm {FONT: [#1] -- \_string#2 \_detokenize{(#3)^^J mods:{#4} vars:{#5} math:{#6}}}%
         \_unless \_ifx #2\_undefined
            \_opwarning{\_string#2 is redefined by \_string\_famdecl\_space[#1]}\_fi
         \_protected\_edef#2{\_def\_noexpand\_currfamily{\_csstring #2}\_unexpanded{#8\_resetfam}}%
         \_ea \_let \_csname _f:\_currfamily:main.fam\_endcsname =#2%
      \_fi
   \_else \_csname _f:\_csstring#2:main.fam\_endcsname \_rm \_endinput \_empty\_fi
}
\_def\_initfontfamily{%
   \_csname _f:\_currfamily:main.fam\_endcsname \_rm
}
```

`\_fvars` ⟨*rm-template*⟩ ⟨*bf-template*⟩ ⟨*it-template*⟩ ⟨*bi-template*⟩ saves data for usage by the `\_currV` macro. If a template is only dot then previous template is used (it can be used if the font family doesn't dispose with all standard variants).
`\_currV` expands to a template declared by `\_fvars` depending on the ⟨*variant name*⟩. Usable only of

standard four variants. Next variants can be declared by the \famvardef macro.
fsetV ⟨*key*⟩=⟨*value*⟩,...,⟨*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 is defined as ⟨*value-a*⟩ or ⟨*value-b*⟩ or ... or ⟨*value-z*⟩.
_prepcommalist ab,{},cd,_fin, expands to ab,,cd, (auxiliary macro used in _onlyif).
_ffonum is a shortcut for oldstyle digits font features used in font family files. You can do \let_ffonum=\ignoreit if you don't want to set old digits together with \caps.

fonts-select.opm

```
\_def\_fvars #1 #2 #3 #4 {%
   \_sdef{_fvar:rm}{#1}%
   \_sdef{_fvar:bf}{#2}%
   \_ifx.#2\_slet{_fvar:bf}{_fvar:rm}\_fi
   \_sdef{_fvar:it}{#3}%
   \_ifx.#3\_slet{_fvar:it}{_fvar:rm}\_fi
   \_sdef{_fvar:bi}{#4}%
   \_ifx.#4\_slet{_fvar:bi}{_fvar:it}\_fi
}
\_def\_currV{\_trycs{_fvar:\_famv}{rm}}
\_def\_V{ }
\_def \_fsetV #1 {\_fsetVa #1,=,}
\_def \_fsetVa #1=#2,{\_isempty{#1}\_iffalse
   \_ifx,#1\_else\_sdef{_#1V}{#2}\_ea\_ea\_ea\_fsetVa\_fi\_fi
}
\_def \_onlyif #1=#2:#3{%
    \_edef\_act{\_noexpand\_isinlist{,\_prepcommalist #2,\_fin,}{,\_cs{_#1V},}}\_act
    \_iftrue #3\_fi
}
\_def\_prepcommalist#1,{\_ifx\_fin#1\_empty\_else #1,\_ea\_prepcommalist\_fi}
\_def\_ffonum {+onum;+pnum}
```

The \moddef \⟨*modifier*⟩ {⟨*data*⟩} simply speaking does \def\⟨*modifier*⟩{⟨*data*⟩}, but we need to respect the family context. In fact, \protected\def_f:⟨*current family*⟩:⟨*modifier*⟩{⟨*data*⟩} is performed and the \⟨*modifier*⟩ is defined as _famdepend\⟨*modifier*⟩{_f:_currfamily:⟨*modifier*⟩}. It expands to _f:_currfamily:⟨*modifier*⟩ value if it is defined or it prints the warning. When the _currfamily value is changed then we can declare the same \⟨*modifier*⟩ with a different meaning.

_setnewmeaning ⟨*cs-name*⟩=_tmpa ⟨*by-what*⟩ does exactly _let ⟨*csname*⟩=_tmpa but warning is printed if ⟨*cs-name*⟩ is defined already and it is not a variant selector or font modifier.

_addtomodlist ⟨*font modifier*⟩ adds given modifier to _modlist macro. This list is used after \resetmod when a new family is selected by a family selector, see _resetfam macro. This allows reinitializing the same current modifiers in the font context after the family is changed.

fonts-select.opm

```
\_def \_moddef #1#2{%
   \_edef\_tmp{\_csstring#1}%
   \_sdef{_f:\_currfamily:\_tmp}{\_addtomodlist#1#2}%
   \_protected \_edef \_tmpa{\_noexpand\_famdepend\_noexpand#1{_f:\_noexpand\_currfamily:\_tmp}}%
   \_setnewmeaning #1=\_tmpa \moddef
}
\_protected \_def\_resetmod {\_cs{_f:\_currfamily:resetmod}} % private variant of \resetmod
\_def \_resetfam{%
   \_def\_addtomodlist##1{}\_resetmod
   \_edef \_modlist{\_ea}\_modlist
   \_let\_addtomodlist=\_addtomodlistb
   \_ifcsname _f:\_currfamily:\_ea\_csstring \_currvar \_endcsname
   \_else \_ea\_ifx\_currvar\_tt \_else \_def\_currvar{\_fontsel}\_fi
   \_fi % corrected \_currvar in the new family
}
\_def \_currfamily{} % default current family is empty
\_def \_modlist{}    % list of currently used modifiers

\_def \_addtomodlist#1{\_addto\_modlist#1}
\_let \_addtomodlistb=\_addtomodlist

\_def\_famdepend#1#2{\_ifcsname#2\_endcsname \_csname#2\_ea\_endcsname \_else
   \_ifx\_addtomodlist\_addtomodlistb
      \_opwarning{\_string#1 is undeclared in family "\_currfamily", ignored}\_fi\_fi
}
```

```
\_def\_setnewmeaning #1=\_tmpa#2{%
   \_ifx #1\_undefined \_else \_ifx #1\_tmpa \_else
      \_opwarning{\_string#1 is redefined by \_string#2}%
   \_fi\_fi
   \_let#1=\_tmpa
}
\_public \moddef ;
```

`\fontdef` ⟨*font-switch*⟩ `{`⟨*data*⟩`}` does:

> `\begingroup` ⟨*data*⟩ `\ea\endgroup \ea\let \ea`⟨*font-switch*⟩ `\the\font`

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

`fonts-select.opm`

```
\_protected\_def \_fontdef #1#2{\_begingroup
   \_edef\_xfamv{\_csstring#1}\_let\_ttfamv\_xfamv #2%
   \_ea\_endgroup\_ea \_let\_ea #1\_the\_font
}
\_public \fontdef ;
```

The `\famvardef` `\xxx` `{`⟨*data*⟩`}` does, roughly speaking:

> `\def \xxx {{`⟨*data*⟩`\ea}\the\font \def\_currvar{\xxx}}`

but the macro `\xxx` is declared as family-dependent. It is analogically as in `\moddef`. The `\xxx` is defined as `\_famdepend\xxx{_f:\_currfamily:xxx}` and `\_f:`⟨*currfam*⟩`:xxx` is defined as mentioned. `\famvardef\tt` behaves somewhat differently: it defines internal version `\_tt` (it is used in `\_ttfont` and `\_urlfont`) and set `\tt` to the same meaning.

`fonts-select.opm`

```
\_protected\_def \_famvardef #1#2{%
   \_sdef{_f:\_currfamily:\_csstring#1}%
      {{\_edef\_xfamv{\_csstring#1}\_let\_ttfamv\_xfamv #2\_ea}\_the\_font \_def\_currvar{#1}}%
   \_protected\_edef\_tmpa {%
      \_noexpand\_famdepend\_noexpand#1{_f:\_noexpand\_currfamily:\_csstring#1}}%
   \_ifx #1\tt
      \_protected\_def\_tt{{\_def\_xfamv{tt}#2\_ea}\_the\_font \_def\_currvar{\_tt}\_matt}%
      \_let\tt=\_tt
   \_else \_setnewmeaning #1=\_tmpa \famvardef
   \_fi
}
\_public \famvardef ;
```

The `\fontfam` `[`⟨*Font Family*⟩`]` does:

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

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

`fonts-select.opm`

```
\_protected\_def \_fontfam [#1]{%
   \_lowercase{\_edef\_famname{\_ea\_removespaces \_expanded{#1} {} }}%
   \_isfile {f-\_famname.opm}\_iftrue \_edef\_famfile{f-\_famname}\_opinput {f-\_famname.opm}%
   \_else
       \_tryloadfamslocal
       \_edef\_famfile{\_trycs{_famf:\_famname}{}}%
       \_ifx\_famfile\_empty
           \_ifcsname _fams:f-\_famname \_endcsname \_edef\_famfile{f-\_famname}\_famsubstitute
           \_else \_listfamnames
           \_fi
       \_else \_opinput {\_famfile.opm}%
   \_fi\_empty\_fi
}
\_def\_tryloadfamslocal{%
   \_isfile {fams-local.opm}\_iftrue
      \_opinput {fams-local.opm}\_famfrom={}\_famsrc={}%
   \_fi
   \_let \_tryloadfamslocal=\_relax  % need not to load fams-local.opm twice
}
\_def\_listfamnames {%
   \_wterm{===== List of font families ======}
   \_begingroup
       \_let\_famtext=\_wterm
       \_def\_faminfo [##1]##2##3##4{%
           \_wterm{ \_space\_noexpand\fontfam [##1] -- ##2}%
       \_let\_famalias=\_famaliasA}%
       \_opinput {fams-ini.opm}%
       \_isfile {fams-local.opm}\_iftrue \_opinput {fams-local.opm}\_fi
       \_message{^^J}%
   \_endgroup
}
\_def\_famaliasA{\_message{ \_space\_space\_space\_space -- alias:}
   \_def\_famalias[##1]{\_message{[##1]}}\_famalias
}
\_public \fontfam ;
```

`\fontfamsub` `[`⟨*Family*⟩`]` `[`⟨*byFamily*⟩`]` declares automatic substitution of ⟨*Family*⟩ by ⟨*byFamily*⟩ which is done when ⟨*Family*⟩ is not installed. I.e. if there is no `f-`⟨*family*⟩`.opm` file or there is no regular font of the family installed. `\_famsubstitute` is internal macro used in `\fontfam` and `\_famdecl` macros. It consumes the rest of the macro, runs `\nospacefuturelet` in order to do `\endinput` to the current `f-file` and runs `\fontfam` again. The table of such substitutions are saved in the macros `\_fams:`⟨*family-file*⟩.

fonts-select.opm

```
\_def \_fontfamsub [#1]#2[#3]{\_tryloadfamslocal
   \_lowercase{\_edef\_tmp{\_removespaces #1 {} }}%
   \_sxdef{_fams:\_trycs{_famf:\_tmp}{f-\_tmp}}{#3}%
}
\_def\_famsubstitute #1\_empty\_fi{\_fi\_fi\_fi
   \_wterm {FONT-SUB: \_famfile\_space -> [\_cs{_fams:\_famfile}]}%
   \_nospacefuturelet\_tmp\_famsubstituteA  % we want to \endinput current f-file
}
\_def\_famsubstituteA{\_fontfam[\_cs{_fams:\_famfile}]}

\_public \fontfamsub ;
```

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

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

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

`\_famalias` `[`⟨*Family Alias*⟩`]` does `\def` `\_famf:`⟨*familyalias*⟩ `{`⟨*file-name*⟩`}` where ⟨*file-name*⟩ is stored from the previous `\_faminfo` command. Or prints information on the terminal.

`\_famfrom` declares type foundry or owner or designer of the font family. It can be used in `fams-ini.opm` or `fams-local.opm` and it is printed in the font catalog.

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

fonts-select.opm

```
\_def\_famtext #1{}
\_def\_faminfo [#1]#2#3#4{%
   \_lowercase{\_edef\_tmp{\_ea\_removespaces #1 {} }}%
   \_edef\_tmpa{f-\_tmp}\_def\_famfile{#3}%
   \_unless\_ifx\_tmpa\_famfile \_sdef{_famf:\_tmp}{#3}\_fi
}
\_def\_famalias [#1]{%
```

```
   \_lowercase{\_edef\_tmpa{\_ea\_removespaces #1 {} }}%
   \_sdef{_famf:\_tmpa\_ea}\_ea{\_famfile}%
}
\_newtoks\_famfrom  \_newtoks\_famsrc
\_input fams-ini.opm
\_let\_famfile=\_undefined
\_famfrom={} \_famsrc={}
```

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

fonts-select.opm

```
\_newtoks \_catalogsample
\_newtoks \_catalogmathsample
\_newtoks \_catalogonly
\_newtoks \_catalogexclude
\_newtoks \_catalognextfam
\_catalogsample={ABCDabcd Qsty fi fl áéíóúüů řžč ÁÉÍÓÚ ŘŽČ 0123456789}
\_catalognextfam={\_bigskip}

\_public \catalogonly \catalogexclude \catalogsample \catalogmathsample \catalognextfam ;
```

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

- `\_defaultfontfeatures` – used for each font,
- `\_ffadded` – features added by `\setff`,
- `\_ffcolor` – features added by `\setfontcolor` (this is obsolette)
- `\_ffletterspace` – features added by `\setletterspace`,
- `\_ffwordspace` – features added by `\setwordspace`.

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

fonts-select.opm

```
\_def \_fontfeatures{\_defaultfontfeatures\_ffadded\_ffcolor\_ffletterspace\_ffwordspace}
\_def \_defaultfontfeatures {+tlig;}
\_def \_ffadded{}
\_def \_ffcolor{}
\_def \_ffletterspace{}
\_def \_ffwordspace{}
```

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

fonts-select.opm

```
\_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 `\fondimen` registers first. It is done by `\_sws:`⟨*fontname*⟩ macro which keeps the original values of the `\fontdimen`s.
`\setfontcolor` is kept here only for backward compatibility but not recommended. Use real color switches and the `\transparency` instead.

fonts-select.opm

```
\_def \_setfontcolor #1{%
   \_edef\_tmp{\_calculatefontcolor{#1}}%
   \_ifx\_tmp\_empty \_def\_ffcolor{}\_else \_edef\_ffcolor{color=\_tmp;}\_fi
}
\_def \_setletterspace #1{%
   \_if^#1^\_def\_ffletterspace{}\_else \_edef\_ffletterspace{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{\_ifx/#1\_ea\_setwspB \_else\_afterfi{\_setwspC#1}\_fi}
\_def\_setwspB #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 ...
\_sdef{_fc:red}{FF0000FF}    \_sdef{_fc:green}{00FF00FF} \_sdef{_fc:blue}{0000FFFF}
\_sdef{_fc:yellow}{FFFF00FF} \_sdef{_fc:cyan}{00FFFFFF}  \_sdef{_fc:magenta}{FF00FFFF}
\_sdef{_fc:white}{FFFFFFFF}  \_sdef{_fc:grey}{00000080}  \_sdef{_fc:lgrey}{00000025}
\_sdef{_fc:black}{} % ... 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:`⟨*templateA*⟩`=\_oz:`⟨*templateB*⟩.

fonts-select.opm

```
\_def\_regoptsizes #1 #2?#3 #4*{\_sdef{_oz:#1}{#2?#3 #4* }}
\_def\_optname #1{\_ifcsname _oz:#1\_endcsname
   \_ea\_ea\_ea \_optnameA \_csname _oz:#1\_ea\_endcsname
   \_else \_failedoptname{#1}\_fi
}
\_def\_failedoptname #1{optname-fails:(#1)}
\_def\_optnameA #1?#2 #3 <#4 {\_ifx*#4#1#3#2\_else
   \_ifdim\_optsize<#4pt #1#3#2\_optnameC
   \_else \_afterfifi \_optnameA #1?#2 \_fi\_fi
}
\_def\_optnameC #1* {\_fi\_fi}
\_def\_afterfifi #1\_fi\_fi{\_fi\_fi #1}
\_def\_optfontalias #1#2{\_slet{_oz:#1}{_oz:#2}}

\_setfontsize {at10pt} % default font size
```

2.14 Preloaded fonts for math mode

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

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

The general rule of usage the math fonts in different sizes in OpTeX says: set three sizes by the macro `\setmathsizes` `[`⟨*text-size*⟩`/`⟨*script-size*⟩`/`⟨*scriptscript-size*⟩`]` and then load all math fonts in given sizes by `\normalmath` or `\boldmath` macros. For example

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

math-preload.opm

```
\_codedecl \normalmath {Math fonts CM + AMS preloaded <2022-12-01>} % preloaded in format
```

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.

math-preload.opm

```
\_def\_normalmath{%
  \_loadmathfamily 0 cmr  % CM Roman
  \_loadmathfamily 1 cmmi % CM Math Italic
  \_loadmathfamily 2 cmsy % CM Standard symbols
  \_loadmathfamily 3 cmex % CM extra symbols
  \_loadmathfamily 4 msam % AMS symbols A
  \_loadmathfamily 5 msbm % AMS symbols B
  \_loadmathfamily 6 rsfs % script
  \_loadmathfamily 7 eufm % fractur
  \_loadmathfamily 8 bfsans % sans serif bold
  \_loadmathfamily 9 bisans % sans serif bold slanted (for vectors)
%  \_setmathfamily 10 \_tentt
%  \_setmathfamily 11 \_tenit
  \_setmathdimens
}
\_def\_boldmath{%
  \_loadmathfamily 0 cmbx  % CM Roman Bold Extended
  \_loadmathfamily 1 cmmib % CM Math Italic Bold
  \_loadmathfamily 2 cmbsy % CM Standard symbols Bold
  \_loadmathfamily 3 cmexb % CM extra symbols Bold
  \_loadmathfamily 4 msam  % AMS symbols A (bold not available?)
  \_loadmathfamily 5 msbm  % AMS symbols B (bold not available?)
  \_loadmathfamily 6 rsfs  % script (bold not available?)
  \_loadmathfamily 7 eufb  % fractur bold
  \_loadmathfamily 8 bbfsans % sans serif extra bold
  \_loadmathfamily 9 bbisans % sans serif extra bold slanted (for vectors)
%  \_setmathfamily 10 \_tentt
%  \_setmathfamily 11 \_tenbi
  \_setmathdimens
}
\_def \normalmath {\_normalmath}  \_def\boldmath {\_boldmath}
```

The classical math family selectors `\mit`, `\cal`, `\bbchar`, `\frak` and `\script` are defined here. The `\rm`, `\bf`, `\it`, `\bi` and `\tt` does two things: they are variant selectors for text fonts and math family selectors for math fonts. The idea was adapted from plain TeX.
These macros are redefined when `unimat-codes.opm` is loaded, see the section 2.16.2.

`math-preload.opm`

```
\_chardef\_bffam = 8
\_chardef\_bifam = 9
%\_chardef\_ttfam = 10
%\_chardef\_itfam = 11

\_protected\_def \_marm {\_fam0 }
\_protected\_def \_mabf {\_fam\_bffam}
\_protected\_def \_mait {\_fam1 }
\_protected\_def \_mabi {\_fam\_bifam}
\_protected\_def \_matt {}

\_protected\_def \_mit    {\_fam1 }
\_protected\_def \_cal    {\_fam2 }
\_protected\_def \_bbchar {\_fam5 }  % double stroked letters
\_protected\_def \_frak   {\_fam7 }  % fraktur
\_protected\_def \_script {\_fam6 }  % more extensive script than \cal

\_public \mit \cal \bbchar \frak \script ;
```

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

`math-preload.opm`

```
%% CM math fonts, optical sizes:

\_regtfm cmmi 0 cmmi5 5.5 cmmi6 6.5 cmmi7 7.5 cmmi8 8.5 cmmi9 9.5
                cmmi10 11.1 cmmi12 *
\_regtfm cmmib 0 cmmib5 5.5 cmmib6 6.5 cmmib7 7.5 cmmib8 8.5 cmmib9 9.5 cmmib10 *
\_regtfm cmtex 0 cstex8 8.5 cstex9 9.5 cstex10 *
\_regtfm cmsy 0 cmsy5 5.5 cmsy6 6.5 cmsy7 7.5 cmsy8 8.5 cmsy9 9.5 cmsy10 *
\_regtfm cmbsy 0 cmbsy5 5.5 cmbsy6 6.5 cmbsy7 7.5 cmbsy8 8.5 cmbsy9 9.5 cmbsy10 *
\_regtfm cmex 0 cmex7 7.5 cmex8 8.5 cmex9 9.5 cmex10 *
\_regtfm cmexb 0 cmexb10 *

\_regtfm cmr  0 cmr5 5.5 cmr6 6.5 cmr7 7.5 cmr8 8.5 cmr9 9.5
                cmr10 11.1 cmr12 15 cmr17 *
\_regtfm cmbx 0 cmbx5 5.5 cmbx6 6.5 cmbx7 7.5 cmbx8 8.5 cmbx9 9.5
                cmbx10 11.1 cmbx12 *
```

```
\_regtfm cmti 0 cmti7 7.5 cmti8 8.5 cmti9 9.5 cmti10 11.1 cmti12 *
\_regtfm cmtt 0 cmtt8 8.5 cmtt9 9.5 cmtt10 11.1 cmtt12 *

%% AMS math fonts, optical sizes:

\_regtfm msam 0 msam5 5.5 msam6 6.5 msam7 7.5 msam8 8.5 msam9 9.5 msam10 *
\_regtfm msbm 0 msbm5 5.5 msbm6 6.5 msbm7 7.5 msbm8 8.5 msbm9 9.5 msbm10 *

%% fraktur, rsfs, optical sizes:

\_regtfm eufm 0 eufm5 6 eufm7 8.5 eufm10 *
\_regtfm eufb 0 eufb5 6 eufb7 8.5 eufb10 *
\_regtfm rsfs 0 rsfs5 6 rsfs7 8.5 rsfs10 *

%% bf and bi sansserif math alternatives:

\_regtfm bfsans 0 ecsx0500 5.5 ecsx0600 6.5 ecsx0700 7.5 ecsx0800
             8.5 ecsx0900 9.5 ecsx1000 11.1 ecsx1200 *
\_regtfm bisans 0 ecso0500 5.5 ecso0600 6.5 ecso0700 7.5 ecso0800
             8.5 ecso0900 9.5 ecso1000 11.1 ecso1200 *
\_regtfm bbfsans 0 ecsx0500 5.5 ecsx0600 6.5 ecsx0700 7.5 ecsx0800
             8.5 ecsx0900 9.5 ecsx1000 11.1 ecsx1200 *
\_regtfm bbisans 0 ecso0500 5.5 ecso0600 6.5 ecso0700 7.5 ecso0800
             8.5 ecso0900 9.5 ecso1000 11.1 ecso1200 *
```

`\_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` ⟨*number*⟩ `\`⟨*font-switch*⟩ loads one math family like `\_loadmathfamily` does it. But the second parameter is a `\`⟨*font-switch*⟩ declared previously by the `\font` primitive.
The ⟨*number*⟩ is saved by `\_loadmathfamily`, `\_setmathfamily` to the `\_mfam`.
The font family is loaded at `\_sizemtext`, `\_sizemscript` and `\_sizemsscript` sizes. These sizes are set by the `\setmathsizes` `[`⟨*text-size*⟩`/`⟨*script-size*⟩`/`⟨*scriptscript-size*⟩`]` macro. These parameters are given in the `\ptmunit` unit, it is set to 1`\ptunit` and it is set to 1 pt by default.
`\_mfactor` sets scaling factor for given math fonts family related to text font size. It does the setting `\_ptmunit=`⟨*factor*⟩`\_ptunit` where the ⟨*factor*⟩ is defined by `\sdef{_mfactor:`⟨*family*⟩`}{`⟨*factor*⟩`}`. 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.

math-preload.opm

```
\_def\_loadmathfamily {\_afterassignment\_loadmathfamilyA \_chardef\_mfam}
\_def\_loadmathfamilyA #1 {\_mfactor
  \_edef\_optsizesave{\_the\_optsize}%
  \_optsize=\_sizemtext    \_font\_mF \_optfn{#1} at\_optsize \_textfont\_mfam=\_mF
  \_optsize=\_sizemscript  \_font\_mF \_optfn{#1} at\_optsize \_scriptfont\_mfam=\_mF
  \_optsize=\_sizemsscript \_font\_mF \_optfn{#1} at\_optsize \_scriptscriptfont\_mfam=\_mF
  \_optsize=\_optsizesave
}
\_def\_setmathfamily {\_afterassignment\_setmathfamilyA \_chardef\_mfam}
\_def\_setmathfamilyA #1{\_mfactor \_let\_mF=#1%
  \_edef\_optsizesave{\_the\_optsize}%
  \_optsize=\_sizemtext    \_fontlet#1#1at\_optsize \_textfont\_mfam=#1%
  \_optsize=\_sizemscript  \_fontlet#1#1at\_optsize \_scriptfont\_mfam=#1%
  \_optsize=\_sizemsscript \_fontlet#1#1at\_optsize \_scriptscriptfont\_mfam=#1%
  \_optsize=\_optsizesave \_let#1=\_mF
}
\_def\_setmathsizes[#1/#2/#3]{\_ptmunit=\_ptunit
   \_def\_sizemtext{#1\_ptmunit}\_def\_sizemscript{#2\_ptmunit}%
   \_def\_sizemsscript{#3\_ptmunit}%
}
\_def\_mfactor{\_ptmunit=\_trycs{_mfactor:\_the\_mfam}{}\_ptunit}

\_newdimen\_ptunit   \_ptunit=1pt
\_newdimen\_ptmunit  \_ptmunit=1\_ptunit

\_public \setmathsizes \ptunit ;
```

`\_setmathparam`⟨*luatex-param*⟩ `{`⟨*factor*⟩`}` sets ⟨*luatex-param*⟩ (like `\Umathspaceafterscript`) to values dependent on `1em` of textfont, scriptfont, scriptscriptfont. The ⟨*factor*⟩ is scaling factor of mentioned `1em`.

math-preload.opm

```
\_def\_setmathparam#1#2{%
   #1\_displaystyle             =#2\_fontdimen6\_textfont1
   #1\_crampeddisplaystyle      =#2\_fontdimen6\_textfont1
   #1\_textstyle                =#2\_fontdimen6\_textfont1
   #1\_crampedtextstyle         =#2\_fontdimen6\_textfont1
   #1\_scriptstyle              =#2\_fontdimen6\_scriptfont1
   #1\_crampedscriptstyle       =#2\_fontdimen6\_scriptfont1
   #1\_scriptscriptstyle        =#2\_fontdimen6\_scriptscriptfont1
   #1\_crampedscriptscriptstyle =#2\_fontdimen6\_scriptscriptfont1
}
```

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

math-preload.opm

```
\_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
  \_skewchar\_scriptscriptfont2=48
  \_skewchar\_textfont6=127 \_skewchar\_scriptfont6=127
  \_skewchar\_scriptscriptfont6=127
}
\_def\_scriptspacefactor{.05}
```

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.

math-preload.opm

```
\_suppressfontnotfounderror=1
\_setmathsizes[10/7/5]
\_ifx\fontspreload\_relax \_else \_normalmath \_fi
\_suppressfontnotfounderror=0
```

2.15 Math macros

math-macros.opm

```
\_codedecl \sin {Math macros plus mathchardefs <2025-04-13>} % preloaded in format
```

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

There is a problem: The `x_n` is tokenized as `x`, `_`, `n` and it works without problems. But `\int_a^b` is tokenized as `\int_a`, `^`, `b`. The control sequence `\int_a` 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 `\`⟨*word*⟩`_` or `\`⟨*word*⟩`_`⟨*one-letter*⟩ (where ⟨*word*⟩ is a sequence of letters) because such control sequences are inaccessible: preprocessor rewrites it.

The `\mathsbon` macro activates the rewriting rule `\`⟨*word*⟩`_`⟨*nonleter*⟩ to `\`⟨*word*⟩ `_`⟨*nonletter*⟩ and `\`⟨*word*⟩`_`⟨*letter*⟩⟨*nonletter*⟩ to `\`⟨*word*⟩ `_`⟨*letter*⟩⟨*nonletter*⟩ 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.

math-macros.opm

```
\catcode`\_ = 8   \let\sb = _
\catcode`\_ = 13  \let _ = \sb
\catcode`\_ = 11
\_private \sb ;

\_newifi\_ifmathsb   \_mathsbfalse
\_def \_mathsbon {%
   \_ifmathsb \_else \_directlua{
   callback.add_to_callback("process_input_buffer",
     function (str)
        local num
        str, num = string.gsub(str.." ", \_gsubrule)
        if num>0 then str = string.gsub(str, \_gsubrule) end % \phi _i\rho _j -> \phi _i\rho _j
        return str
     end, "_mathsb") }\_fi
   \_global\_mathsbtrue
}
\_def \_mathsboff {%
   \_ifmathsb \_directlua{ callback.remove_from_callback("process_input_buffer", "_mathsb") }\_fi
   \_global \_mathsbfalse
}
\_edef\_gsubrule{"(\_nbb[a-zA-Z]+)_([a-zA-Z]?[^_a-zA-Z])", "\_pcent 1 _\_pcent 2"}

\_public \mathsboff \mathsbon ;
```

All mathcodes are set to equal values as in plainTEX. But all encoding-dependent declarations (like these) will be set to different values when a Unicode-math font is used.

math-macros.opm

```
\_mathcode`^^@="2201 % \cdot
\_mathcode`^^A="3223 % \downarrow
\_mathcode`^^B="010B % \alpha
\_mathcode`^^C="010C % \beta
\_mathcode`^^D="225E % \land
\_mathcode`^^E="023A % \lnot
\_mathcode`^^F="3232 % \in
\_mathcode`^^G="0119 % \pi
\_mathcode`^^H="0115 % \lambda
\_mathcode`^^I="010D % \gamma
\_mathcode`^^J="010E % \delta
\_mathcode`^^K="3222 % \uparrow
\_mathcode`^^L="2206 % \pm
\_mathcode`^^M="2208 % \oplus
\_mathcode`^^N="0231 % \infty
\_mathcode`^^O="0140 % \partial
\_mathcode`^^P="321A % \subset
\_mathcode`^^Q="321B % \supset
\_mathcode`^^R="225C % \cap
\_mathcode`^^S="225B % \cup
\_mathcode`^^T="0238 % \forall
\_mathcode`^^U="0239 % \exists
\_mathcode`^^V="220A % \otimes
\_mathcode`^^W="3224 % \leftrightarrow
\_mathcode`^^X="3220 % \leftarrow
\_mathcode`^^Y="3221 % \rightarrow
\_mathcode`^^Z="8000 % \ne
\_mathcode`^^[="2205 % \diamond
\_mathcode`^^"3214 % \le
\_mathcode`^^]="3215 % \ge
\_mathcode`^^^="3211 % \equiv
\_mathcode`^^_="225F % \lor
\_mathcode`="8000 % \space
\_mathcode`="5021
\_mathcode`="8000 % ^\prime
\_mathcode`="4028
\_mathcode`="5029
\_mathcode`="2203 % \ast
\_mathcode`="202B
\_mathcode`="613B
\_mathcode`="2200
```

```
\_mathcode`\.="013A
\_mathcode`="013D
\_mathcode`="303A
\_mathcode`="603B
\_mathcode`="313C
\_mathcode`="303D
\_mathcode`="313E
\_mathcode`="503F
\_mathcode`="405B
\_mathcode`="026E % \backslash
\_mathcode`="505D
\_mathcode`\_="8000 % math-active subscript
\_mathcode`="4266
\_mathcode`="026A
\_mathcode`="5267
\_mathcode`^?="1273 % \smallint

\_delcode`="028300
\_delcode`="029301
\_delcode`="05B302
\_delcode`="05D303
\_delcode`="26830A
\_delcode`="26930B
\_delcode`="02F30E
\_delcode`="26A30C
\_delcode`="26E30F
```

All control sequences declared by `\mathchardef` are supposed (by default) only for public usage. It means that they are declared without _ prefix. If such sequences are used in internal OpTEX macro then their internal prefixed form is declared using `\_private` macro.
These encoding dependent declarations will be set to different values when Unicode-math font is loaded. The declared sequences for math symbols are not hyperlinked in this documentation.

`math-macros.opm`

```
\_mathchardef\alpha="010B
\_mathchardef\beta="010C
\_mathchardef\gamma="010D
\_mathchardef\delta="010E
\_mathchardef\epsilon="010F
\_mathchardef\zeta="0110
\_mathchardef\eta="0111
\_mathchardef\theta="0112
\_mathchardef\iota="0113
\_mathchardef\kappa="0114
\_mathchardef\lambda="0115
\_mathchardef\mu="0116
\_mathchardef\nu="0117
\_mathchardef\xi="0118
\_mathchardef\pi="0119
```

. . . etc. (see `math-macros.opm`)

The math functions like log, sin, cos are declared in the same way as in plainTEX, but they are `\protected` in OpTEX.

`math-macros.opm`

```
\_protected\_def\log {\_mathop{\_rm log}\_nolimits}
\_protected\_def\lg {\_mathop{\_rm lg}\_nolimits}
\_protected\_def\ln {\_mathop{\_rm ln}\_nolimits}
\_protected\_def\lim {\_mathop{\_rm lim}}
\_protected\_def\limsup {\_mathop{\_rm lim\_thinsk sup}}
\_protected\_def\liminf {\_mathop{\_rm lim\_thinsk inf}}
\_protected\_def\sin {\_mathop{\_rm sin}\_nolimits}
\_protected\_def\arcsin {\_mathop{\_rm arcsin}\_nolimits}
\_protected\_def\sinh {\_mathop{\_rm sinh}\_nolimits}
\_protected\_def\cos {\_mathop{\_rm cos}\_nolimits}
\_protected\_def\arccos {\_mathop{\_rm arccos}\_nolimits}
\_protected\_def\cosh {\_mathop{\_rm cosh}\_nolimits}
\_protected\_def\tan {\_mathop{\_rm tan}\_nolimits}
\_protected\_def\arctan {\_mathop{\_rm arctan}\_nolimits}
\_protected\_def\tanh {\_mathop{\_rm tanh}\_nolimits}
```

```
\_protected\_def\cot {\_mathop{\_rm cot}\_nolimits}
\_protected\_def\coth {\_mathop{\_rm coth}\_nolimits}
%\_protected\_def\sec {\_mathop{\_rm sec}\_nolimits} % \sec is section
\_protected\_def\secant {\_mathop{\_rm sec}\_nolimits}
\_protected\_def\csc {\_mathop{\_rm csc}\_nolimits}
\_protected\_def\max {\_mathop{\_rm max}}
\_protected\_def\min {\_mathop{\_rm min}}
\_protected\_def\sup {\_mathop{\_rm sup}}
\_protected\_def\inf {\_mathop{\_rm inf}}
\_protected\_def\arg {\_mathop{\_rm arg}\_nolimits}
\_protected\_def\ker {\_mathop{\_rm ker}\_nolimits}
\_protected\_def\dim {\_mathop{\_rm dim}\_nolimits}
\_protected\_def\hom {\_mathop{\_rm hom}\_nolimits}
\_protected\_def\det {\_mathop{\_rm det}}
\_protected\_def\exp {\_mathop{\_rm exp}\_nolimits}
\_protected\_def\Pr {\_mathop{\_rm Pr}}
\_protected\_def\gcd {\_mathop{\_rm gcd}}
\_protected\_def\deg {\_mathop{\_rm deg}\_nolimits}
```

These macros are defined similarly as in plainTeX. Only internal macro names from plainTeX with `@` character are re-written in a more readable form.
`\sp` is an alternative for `^`. The `\sb` alternative for `_` was defined at line 27 of the file `math-macros.opm`.
`\_thinsk`, `\_medsk`, `\_thicksk` and `\_thinneg` should be used instead `\,`, `\>`, `\;` and `\!` in macros because a user can re-define these single-letter sequences.

`math-macros.opm`

```
\_let\_sp=^ \public \sp ;
% \sb=_ , defined at beginning of this file

\_def\_thinsk {\_mskip\_thinmuskip}
\_protected\_def{\_relax\_ifmmode \_thinsk \_else \_thinspace \_fi}
\_protected\_def{\_mskip\_medmuskip}    \let\_medsk   =
\_protected\_def{\_mskip\_thickmuskip}  \let\_thicksk =
\_protected\_def{\_mskip-\_thinmuskip}  \let\_thinneg =
%\_def{\discretionary{\thinspace\the\textfont2\char2}{}{}} % obsolete
```

Active `\prime` character is defined here.

`math-macros.opm`

```
{\_catcode`=\active \_gdef'{^\_bgroup\_primes}} % primes dance
\_def\_primes{\_prime\_isnextchar'{\_primesA}%
                                  {\_isnextchar^{\_primesB}{\_egroup}}}
\_def\_primesA #1{\_primes}
\_def\_primesB #1#2{#2\egroup}
\_private \prime ;
```

`\big`, `\bbig`, `\Big`, `\bBig`, `\bigg`, `\Bigg`, `\bigl`, `\bigm`, `\bigr`, `\bbigl`, `\bbigm`, `\bbigr`, `\bBigl`, `\Bigl`, `\Bigm`, `\bBigm`, `\Bigr`, `\bBigr`, `\biggl`, `\biggm`, `\biggr`, `\Biggl`, `\Biggm`, `\Biggr` are based on the `\_scalebig`⟨*fence*⟩⟨*num*⟩`;` macro because we need the dependency on the various sizes of the fonts. The `\_scalebigcoef`⟨*num*⟩ returns relevant coefficient for these macros. Multiply this coefficient by two and you get the strut height+depth in em units. If the ⟨*num*⟩ is bigger than 6 then the next parameter is used directly, for example `\def\morebig#1{\_scalebig{#1}7{1.8}}`
The `\big`, `\Big`, `\bigg`, `\Bigg` macros keep the strut height+depth from plain TeX and `\bbig` is a new macro in OpTeX. It generates the size 1.44 em between `\big` and `\Big` which is accessible in most of Unicode math fonts (but not in classical `cmex10`).

`math-macros.opm`

```
%{\catcode`^^Z=\active \gdef^^Z{\not=}} % ^^Z is like \ne in math %obsolete

\_def\_scalebig#1#2#3{{\_left#1%
    \_raise\_Umathaxis\_textstyle\_vbox to\_scalebigcoef{#2}{#3}\_fontdimen6\_textfont1{}%
    \_kern-\_nulldelimiterspace\_right.}}
\_def\_scalebigcoef#1#2{\_ifcase #1 0\_or
% \big (1.22) \bbig (1.44) \Big (1.8) \Bibg (2.1) \bigg (2.4) \Bigg (3.0)  % \_scalebig7{scale}
         .62\_or      .72\_or     .9\_or     1.05\or     1.2\_or     1.5\_else #2\_fi}
\_protected\_def\_big #1{\_scalebig{#1}1;}
\_protected\_def\_bbig#1{\_scalebig{#1}2;}
\_protected\_def\_Big #1{\_scalebig{#1}3;}
\_protected\_def\_bBig#1{\_scalebig{#1}4;}
\_protected\_def\_bigg#1{\_scalebig{#1}5;}
```

```
\_protected\_def\_Bigg#1{\_scalebig{#1}6;}
\_public \big \bbig \Big \bBig \bigg \Bigg ; % only \big \Big \bigg \Bigg in original plain TeX

\_protected\_def\_bigl{\_mathopen\_big}
\_protected\_def\_bigm{\_mathrel\_big}
\_protected\_def\_bigr{\_mathclose\_big}
\_protected\_def\_bbigl{\_mathopen\_bbig}
\_protected\_def\_bbigm{\_mathrel\_bbig}
\_protected\_def\_bbigr{\_mathclose\_bbig}
\_protected\_def\_Bigl{\_mathopen\_Big}
\_protected\_def\_bBigl{\_mathopen\_bBig}
\_protected\_def\_Bigm{\_mathrel\_Big}
\_protected\_def\_bBigm{\_mathrel\_bBig}
\_protected\_def\_Bigr{\_mathclose\_Big}
\_protected\_def\_bBigr{\_mathclose\_bBig}
\_protected\_def\_biggl{\_mathopen\_bigg}
\_protected\_def\_biggm{\_mathrel\_bigg}
\_protected\_def\_biggr{\_mathclose\_bigg}
\_protected\_def\_Biggl{\_mathopen\_Bigg}
\_protected\_def\_Biggm{\_mathrel\_Bigg}
\_protected\_def\_Biggr{\_mathclose\_Bigg}
\_public \bigl \bigm \bigr \bbigl \bbigm \bbigr
         \Bigl \Bigm \Bigr \bBigl \bBigm \bBigr \biggl \biggm \biggr \Biggl \Biggm \Biggr ;
```

Math relations defined by the \jointrel plain TeX macro:

math-macros.opm

```
\_protected\_def\_joinrel{\_mathrel{\_mkern-2.5mu}}  % -3mu in plainTeX
\_protected\_def\relbar{\_mathrel{\_smash-}} % \_smash, because - has the same height as +
\_protected\_def\Relbar{\_mathrel=}
\_mathchardef\lhook="312C
\_protected\_def\hookrightarrow{\_lhook\_joinrel\_rightarrow}
\_mathchardef\rhook="312D
\_protected\_def\hookleftarrow{\_leftarrow\_joinrel\_rhook}
\_protected\_def\bowtie{\_mathrel\_triangleright\_joinrel\_mathrel\_triangleleft}
\_protected\_def\models{\_mathrel|\_joinrel=}
\_protected\_def\Longrightarrow{\_Relbar\_joinrel\_Rightarrow}
\_protected\_def\longrightarrow{\_relbar\_joinrel\_rightarrow}
\_protected\_def\longleftarrow{\_leftarrow\_joinrel\_relbar}
\_protected\_def\Longleftarrow{\_Leftarrow\_joinrel\_Relbar}
\_protected\_def\longmapsto{\_mapstochar\_longrightarrow}
\_protected\_def\longleftrightarrow{\_leftarrow\_joinrel\_rightarrow}
\_protected\_def\Longleftrightarrow{\_Leftarrow\_joinrel\_Rightarrow}
\_protected\_def\iff{\_thicksk\_Longleftrightarrow\_thicksk}
\_private \lhook \rightarrow \leftarrow \rhook \triangleright \triangleleft
   \Relbar \Rightarrow \relbar \rightarrow \Leftarrow \mapstochar
   \longrightarrow \Longleftrightarrow ;
\_public \joinrel ;
```

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

math-macros.opm

```
\_mathchardef\_ldotp="613A % ldot as a punctuation mark
\_mathchardef\_cdotp="6201 % cdot as a punctuation mark
\_mathchardef\_colon="603A % colon as a punctuation mark
\_public \ldotp \cdotp \colon ;

\_protected\_def\_ldots{\_mathinner{\_ldotp\_ldotp\_ldotp}}
\_protected\_def\_cdots{\_mathinner{\_cdotp\_cdotp\_cdotp}}
\_protected\_def\_vdots{\_vbox{\_baselineskip=.4em \_lineskiplimit=\_zo
    \_kern.6em \_hbox{.}\_hbox{.}\_hbox{.}}}
\_protected\_def\_ddots{\_mathinner{%
    \_mkern1mu\_raise.7em\_vbox{\_kern.7em\_hbox{.}}\_mkern2mu
    \_raise.4em\_hbox{.}\_mkern2mu\_raise.1em\_hbox{.}\_mkern1mu}}

\_public \ldots \cdots \vdots \ddots ;
```

\adots inspired by plain TeX

math-macros.opm

```
\_protected\_def\_adots{\_mathinner{%
    \_mkern1mu\_raise.1em\_hbox{.}\_mkern2mu
    \_raise.4em\_hbox{.}\_mkern2mu\_raise.7em\_vbox{\_kern.7em\_hbox{.}}\_mkern1mu}}
```

```

\_public \adots ;
```

Math accents (encoding dependent declarations).

math-macros.opm

```
\_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.

math-macros.opm

```
\_def\_math{\_mathsurround\_zo}
\_protected\_def\_skew #1#2#3{{\_muskip0=#1mu\_divide\_muskip0by2 \_mkern\_muskip0
    #2{\_mkern-\_muskip0{#3}\_mkern\_muskip0}\_mkern-\_muskip0}{}}
\_protected\_def\_overrightarrow #1{\_vbox{\_math\_ialign{##\_crcr
      \_rightarrowfill\_crcr\_noalign{\_kern-.1em \_nointerlineskip}
      $\_hfil\_displaystyle{#1}\_hfil$\_crcr}}}
\_protected\_def\_overleftarrow #1{\_vbox{\_math\_ialign{##\_crcr
      \_leftarrowfill\_crcr\_noalign{\_kern-.1em \_nointerlineskip}
      $\_hfil\_displaystyle{#1}\_hfil$\_crcr}}}
\_protected\_def\_overbrace #1{\_mathop{%
      \_vbox{\_math\_ialign{##\_crcr\_noalign{\_kern.3em}
      \_downbracefill\_crcr\_noalign{\_kern.3em \_nointerlineskip}
      $\_hfil\_displaystyle{#1}\_hfil$\_crcr}}}\_limits}
\_protected\_def\_underbrace #1{\_mathop{\_vtop{\_math\_ialign{##\_crcr
      $\_hfil\_displaystyle{#1}\_hfil$\_crcr\_noalign{\_kern.3em \_nointerlineskip}
      \_upbracefill\_crcr\_noalign{\_kern.3em}}}}\_limits}

\_public \overrightarrow \overleftarrow \overbrace \underbrace \skew ;
```

Macros based on `\delimiter`, `\*witdelims` and `\radical` primitives.

math-macros.opm

```
\_protected\_def\lmoustache{\_delimiter"437A340 } % top from (, bottom from )
\_protected\_def\rmoustache{\_delimiter"537B341 } % top from ), bottom from (
\_protected\_def\lgroup{\_delimiter"462833A } % extensible ( with sharper tips
\_protected\_def\rgroup{\_delimiter"562933B } % extensible ) with sharper tips
\_protected\_def\arrowvert{\_delimiter"26A33C } % arrow without arrowheads
\_protected\_def\Arrowvert{\_delimiter"26B33D } % double arrow without arrowheads
\_protected\_def\bracevert{\_delimiter"77C33E } % the vertical bar that extends braces
\_protected\_def\Vert{\_delimiter"26B30D } \_let=\Vert
\_protected\_def\vert{\_delimiter"26A30C }
\_protected\_def\uparrow{\_delimiter"3222378 }
\_protected\_def\downarrow{\_delimiter"3223379 }
\_protected\_def\updownarrow{\_delimiter"326C33F }
\_protected\_def\Uparrow{\_delimiter"322A37E }
\_protected\_def\Downarrow{\_delimiter"322B37F }
\_protected\_def\Updownarrow{\_delimiter"326D377 }
\_protected\_def\backslash{\_delimiter"26E30F } % for double coset G\_backslash H
\_protected\_def\langle{\_delimiter"426830A }
\_protected\_def\rangle{\_delimiter"526930B }
\_protected\_def\lbrace{\_delimiter"4266308 } \_let\_lbrace=\lbrace
\_protected\_def\rbrace{\_delimiter"5267309 } \_let\_rbrace=\rbrace
\_protected\_def{\_ifmmode \_lbrace\_else\_char` \_fi}
\_protected\_def{\_ifmmode \_rbrace\_else\_char` \_fi}

\_protected\_def\rceil{\_delimiter"5265307 }
\_protected\_def\lceil{\_delimiter"4264306 }
\_protected\_def\rfloor{\_delimiter"5263305 }
\_protected\_def\lfloor{\_delimiter"4262304 }

```

```
\_protected\_def\choose{\_atopwithdelims()}
\_protected\_def\brack{\_atopwithdelims[]}
\_protected\_def\brace{\_atopwithdelims\_lbrace\_rbrace}

\_protected\_def\_sqrt{\_radical"270370 }  \_public \sqrt ;
```

`\mathpalette`, `\vphantom`, `\hphantom`, `\phantom`, `\mathstrut`, and `\smash` macros from plain TeX.

math-macros.opm

```
\_def\_mathpalette#1#2{\_mathchoice{#1\_displaystyle{#2}}%
  {#1\_textstyle{#2}}{#1\_scriptstyle{#2}}{#1\_scriptscriptstyle{#2}}}
\_newbox\_rootbox
\_protected\_def\root#1\of{\_setbox\_rootbox
  \_hbox{$\_math\_scriptscriptstyle{#1}$}\_mathpalette\_rootA}
\_def\_rootA#1#2{\_setbox0=\_hbox{$\_math#1\_sqrt{#2}$}\_dimen0=\_ht0
  \_advance\_dimen0by-\_dp0
  \_mkern5mu\_raise.6\_dimen0\_copy\_rootbox \_mkern-10mu\_box0 }
\_newifi\_ifvp \_newifi\_ifhp
\_protected\_def\_vphantom{\_vptrue\_hpfalse\_phant}
\_protected\_def\_hphantom{\_vpfalse\_hptrue\_phant}
\_protected\_def\_phantom{\_vptrue\_hptrue\_phant}
\_def\_phant{\_ifmmode\_def\_next{\_mathpalette\_mathphant}%
  \_else\_let\_next=\_makephant\_fi\_next}
\_def\_makephant#1{\_setbox0\_hbox{#1}\_finphant}
\_def\_mathphant#1#2{\_setbox0=\_hbox{$\_math#1{#2}$}\_finphant}
\_def\_finphant{\_setbox2=\_null
  \_ifvp \_ht2=\_ht0 \_dp2=\_dp0 \_fi
  \_ifhp \_wd2=\_wd0 \_fi \_hbox{\_box2}}
\_def\_mathstrut{\_vphantom()}
\_protected\_def\_smash{\_relax % \_relax, in case this comes first in \halign
  \_ifmmode\_def\_next{\_mathpalette\_mathsmash}\_else\_let\_next\_makesmash
  \_fi\_next}
\_def\_makesmash#1{\_setbox0=\_hbox{#1}\_finsmash}
\_def\_mathsmash#1#2{\_setbox0=\_hbox{$\_math#1{#2}$}\_finsmash}
\_def\_finsmash{\_ht0=\_zo \_dp0=\_zo \_hbox{\_box0}}
\_public \mathpalette \vphantom \hphantom \phantom \mathstrut \smash ;
```

`\cong`, `\notin`, `\rightleftharpoons`, `\buildrel`, `\doteq`, `\bmod` and `\pmod` macros from plain TeX.

math-macros.opm

```
\_protected\_def\_cong{\_mathrel{\_mathpalette\_overeq\_sim}} % congruence sign
\_def\_overeq#1#2{\_lower.05em\_vbox{\_lineskiplimit\_maxdimen\_lineskip=-.05em
    \_ialign{$\_math#1\_hfil##\_hfil$\_crcr#2\_crcr=\_crcr}}}
\_protected\_def\_notin{\_mathrel{\_mathpalette\_icancel\_in}}
\_def\_icancel#1#2{\_math\_ooalign{$\_hfil#1\_mkern1mu/\_hfil$\_crcr$#1#2$}}
\_protected\_def\_rightleftharpoons{\_mathrel{\_mathpalette\_rlhp{}}}
\_def\_rlhp#1{\_vcenter{\_math\_hbox{\_ooalign{\_raise.2em
        \_hbox{$#1\_rightharpoonup$}\_crcr
      $#1\_leftharpoondown$}}}}
\_protected\_def\_buildrel#1\over#2{\_mathrel{\_mathop{\_kern\_zo #2}\_limits^{#1}}}
\_protected\_def\_doteq{\_buildrel\_textstyle.\over=}
\_private \in \sim ;
\_public \cong \notin \rightleftharpoons \buildrel \doteq ;

\_protected\_def\_bmod{\_nonscript\_mskip-\_medmuskip\_mkern5mu
  \_mathbin{\_rm mod}\_penalty900\_mkern5mu\_nonscript\_mskip-\_medmuskip}
\_protected\_def\_pmod#1{\_allowbreak\_mkern18mu({\_rm mod}\_thinsk\_thinsk#1)}
\_public \bmod \pmod ;
```

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

math-macros.opm

```
\_protected\_def\_matrix#1{\_null\_thinsk
    \_edef\_tmpa{\_the\_numexpr \_mathstyle/4\_relax}% 0 0 1 1 1 1 2 2
    \_vcenter{\_matrixbaselines\_math
    \_ialign{\_the\_lmfil$\_matrixstyle##$\_hfil&&\_quad\_the\_lmfil$\_matrixstyle##$\_hfil\_crcr
     \_mathstrut\_crcr\_noalign{\_kern-\_baselineskip}
     #1\_crcr\_mathstrut\_crcr\_noalign{\_kern-\_baselineskip}}}\_thinsk}

\_def\_matrixbaselines{\_normalbaselines \_def\_matrixstyle{}%
```

```
   \_let\_matrixbaselines=\_relax % \matrix inside matrix does not change size again
   \_ifcase\_tmpa \_or
       \_baselineskip=.7\_baselineskip \_def\_quad {\_hskip.7em\_relax}%
       \_let\_matrixstyle=\_scriptstyle
   \_or
       \_baselineskip=.5\_baselineskip \_def\_quad {\_hskip.5em\_relax}%
       \_let\_matrixstyle=\_scriptscriptstyle
   \_fi
}
\_protected\_def\_pmatrix#1{\_left(\_matrix{#1}\_right)}

\_public \matrix \pmatrix ;
```

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.

math-macros.opm

```
\_protected\_long\_def\_cases#1{\_left\_thinsk\_vcenter{\_normalbaselines\_math
    \_ialign{$##\_hfil$&\_quad{##\_unsskip}\_hfil\_crcr#1\_crcr}}\_right.}

\_newdimen\_ptrenwd
\_ptrenwd=8.75pt % width of the big left (
\_protected\_def\_bordermatrix{\_bordermatrixwithdelims()}
\_def\_bordermatrixwithdelims#1#2#3{\_begingroup \_math
  \_setbox0=\_vbox{\_bordermatrixA #3\_stopbmatrix}%
  \_setbox2=\_vbox{\_unvcopy0 \_global\_setbox1=\_lastbox}%
  \_setbox2=\_hbox{\_unhbox1 \_unskip\_global\_setbox1=\_lastbox}%
  \_setbox2=\_hbox{$\_kern\_wd1 \_kern-\_ptrenwd\_left#1\_kern-\_wd1
    \_global\_setbox1=\_vbox{\_box1 \_kern.2em}%
    \_vcenter{\_kern-\_ht1 \_unvbox0 \_kern-\_baselineskip}\_thinsk\_right#2$}%
  \_null\_thicksk\_vbox{\_kern\_ht1 \_box2}\_endgroup}
\_def\_bordermatrixA #1\cr#2\_stopbmatrix{%
    \_ialign{$###$\_hfil\_kern.2em\_kern\_ptrenwd&\_thinspace\_hfil$###$\_hfil
      &&\_quad\_hfil$###$\_hfil\_crcr
      \_omit\_strut\_hfil\_crcr\_noalign{\_kern-\_baselineskip}%
      #1\_crcr\_noalign{\_kern.2em}#2\_crcr\_omit\_strut\_cr}}

\_public \cases \bordermatrix ;
```

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

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

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

math-macros.opm

```
\_long\_def\_eqalign#1{\_null\_thinsk\_vcenter{\_the\_eqlines\_math
   \_ialign{&\_hfil$\_the\_eqstyle{##}$&$\_the\_eqstyle{{}##}$\_hfil
            &\_hskip.5\_eqspace\_hfil$\_the\_eqstyle{##}$\_hskip.5\_eqspace\_hfil
            \_crcr#1\_crcr}}\_thinsk}

\_public \eqalign ;
```

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

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

math-macros.opm

```
\_def\_displaylines #1#{\_ifx&#1&\_ea\_displaylinesD
   \_else \_def\_tmp to##1\_end{\_def\_tmp{\_dimexpr ##1}}\_tmp #1\_end
        \_ea\_displaylinesto \_fi}
\_long\_def\_displaylinesD #1{\_display \_tabskip=\_zoskip
```

```
   \_halign{\_hbox to\_displaywidth{$\_elign\_hfil\_displaystyle##\_hfil$}\_crcr
      #1\_crcr}}
\_long\_def\_displaylinesto #1{\_vcenter{\_openup\_jot \_math \_tabskip=\_zoskip
   \_halign{\_strut\_hbox to\_span\_tmp{$\_hss\_displaystyle##\_hss$}\_crcr
      #1\_crcr}}}

\_public\displaylines ;
```

`\openup`, `\eqalignno` and `\leqalignno` macros are copied from Plain TEX unchanged.

math-macros.opm

```
\_def\_openup{\_afterassignment\_openupA\_dimen0=}
\_def\_openupA{\_advance\_lineskip by\_dimen0
  \_advance\_baselineskip by\_dimen0
  \_advance\_lineskiplimit by\_dimen0 }
\_newifi\_ifdtop
\_def\_display{\_global\_dtoptrue\_openup\_jot\_math
  \_everycr{\_noalign{\_ifdtop \_global\_dtopfalse \_ifdim\_prevdepth>-1000pt
      \_vskip-\_lineskiplimit \_vskip\_normallineskiplimit \_fi
      \_else \_penalty\_interdisplaylinepenalty \_fi}}}
\_def\_elign{\_tabskip=\_zoskip\_everycr{}} % restore inside \_display
\_long\_def\_eqalignno#1{\_display \_tabskip=\_centering
  \_halign to\_displaywidth{\_hfil$\_elign\_displaystyle{##}$\_tabskip=\_zoskip
    &$\_elign\_displaystyle{{}##}$\_hfil\_tabskip\_centering
    &\_hbox to\_zo{\_hss$\_elign##$}\_tabskip\_zoskip\_crcr
    #1\_crcr}}
\_long\_def\_leqalignno#1{\_display \_tabskip=\_centering
  \_halign to\_displaywidth{\_hfil$\_elign\_displaystyle{##}$\_tabskip=\_zoskip
    &$\_elign\_displaystyle{{}##}$\_hfil\_tabskip=\_centering
    &\_kern-\_displaywidth\_hbox to\_zo{$\_elign##$\_hss}\_tabskip\_displaywidth\_crcr
    #1\_crcr}}
\_public \openup \eqalignno \leqalignno ;
```

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

math-macros.opm

```
\_def\_amsafam{4} \_def\_amsbfam{5}

\_mathchardef \boxdot   "2\_amsafam 00
\_mathchardef \boxplus   "2\_amsafam 01
\_mathchardef \boxtimes   "2\_amsafam 02
\_mathchardef \square   "0\_amsafam 03
\_mathchardef \blacksquare   "0\_amsafam 04
\_mathchardef \centerdot   "2\_amsafam 05
\_mathchardef \lozenge   "0\_amsafam 06
\_mathchardef \blacklozenge   "0\_amsafam 07
\_mathchardef \circlearrowright   "3\_amsafam 08
\_mathchardef \circlearrowleft   "3\_amsafam 09
\_mathchardef \rightleftharpoons   "3\_amsafam 0A
\_mathchardef \leftrightharpoons   "3\_amsafam 0B
\_mathchardef \boxminus   "2\_amsafam 0C
```

. . . etc. (see math-macros.opm)

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.
```

math-macros.opm

```
\_mathchardef \_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 \npreceq
  \nsucceq \nleqq \ngeqq \nsim \ncong \nsubseteqq \nsupseteqq \nsubseteq
  \nsupseteq \nparallel \nmid \nshortmid \nshortparallel \nvdash \nVdash
  \nvDash \nVDash \ntrianglerighteq \ntrianglelefteq \ntriangleleft
  \ntriangleright \nleftarrow \nrightarrow \nLeftarrow \nRightarrow
  \nLeftrightarrow \nleftrightarrow \nexists ;
\_public \not ;
```

`\mathstyles{`⟨*math list*⟩`}` behaves like `{`⟨*math list*⟩`}`, but you can use the following commands in the ⟨*math list*⟩:

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

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

math-macros.opm

```
\_newcount\_stylenum
\_def\_mathstyles#1{{\_mathchoice{\_stylenum0 #1}{\_stylenum1 #1}%
                                {\_stylenum2 #1}{\_stylenum3 #1}}}
\_def\_dobystyle#1#2#3#4{\_ifcase\_stylenum#1\_or#2\_or#3\_or#4\_fi}
\_def\_currstyle{\_dobystyle\_displaystyle\_textstyle\_scriptstyle\_scriptscriptstyle}
\_public \mathstyles \dobystyle \currstyle \stylenum ;
```

The `\cramped` macro sets the cramped variant of the current style. Note that `\currstyle` initializes non-cramped variants. The example `\mathframe` above should be:
`\def\mathframe#1{\mathstyles{\frame{$\currstyle\cramped #1$}}}`.
Second note: `\cramped` macro reads the current math style from the `\mathstyle` LuaTeX primitive, so it does not work in numerators of generalized fractions but you can use it before the fraction is opened: `$\cramped {x^2\over y^2}$`.

math-macros.opm

```
\_def\_cramped{\_ifcase\_numexpr(\_mathstyle+1)/2\_relax\_or
   \_crampeddisplaystyle \_or \_crampedtextstyle \_or
   \_crampedscriptstyle \_or \_crampedscriptscriptstyle \_fi
}
\_public \cramped ;
```

`\setmathstyle` 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{{\setmathstyle \hbox to0pt{\hss$\usemathstyle#1$\hss}}}`.

math-macros.opm

```
\_newcount\_mstylenum
\_def\_setmathstyle{\_mstylenum=\_mathstyle\_relax}
\_def\_usemathstyle{\_ifcase\_mstylenum
   \_displaystyle\_or \_crampeddisplaystyle\_or \_textstyle\_or \_crampedtextstyle\_or
   \_scriptstyle\_or \_crampedscriptstyle\_or \_scriptscriptstyle\_or \_crampedscriptscriptstyle
   \_fi
}
\_public \setmathstyle \usemathstyle ;
```

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

math-macros.opm

```
\_def\_mathbox#1{{\_mathstyles{\_hbox{%
   \_ifnum\_stylenum<2 \_everymath{\_currstyle}%
   \_else \_ifx \_normalmath\_normalunimath
      \_ifnum\_stylenum=2 \_def\_textmff{ssty=1;}\_fi
      \_ifnum\_stylenum=3 \_def\_textmff{ssty=2;}\_def\_scriptmff{ssty=2;}\_fi
      \_typosize[\_ea\_ignorept\_the\_fontdimen6\_dobystyle{}{}\_scriptfont\_scriptscriptfont1/]%
   \_else \_typoscale[\_dobystyle{}{}{700}{500}/]%
   \_fi\_fi #1}}}%
}
\_public \mathbox ;
```

2.16 Unicode-math fonts

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

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

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

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

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

The *main math font* is loaded by `\loadmath` (typically indirectly using `\fontfam`) and you can load more *additional math fonts* by `\addUmathfont`:

```
\addUmathfont \famname {[⟨normal-font⟩]}{⟨ffeatures⟩} {[⟨bold-font⟩]}{⟨ffeatures⟩} {⟨factor⟩}
```

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

```
\fontfam[lm]  % does \lodmath{[latinmodern-math]}
\addUmathfont \xits {[XITSMath-Regular]}{} {[XITSMath-Bold]}{} {}
```

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

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

```
\mathchars \xits {\stareq \triangleq \veeeq \wedgeq}
```

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

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

Use `\addto\selector{\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

```
\Umathcode `⟨character⟩ = ⟨class-number⟩ ⟨family⟩ `⟨character⟩
```

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

2.16.1 Unicode-math macros preloaded in the format

unimath-macros.opm

```
\_codedecl \loadmath {Unicode Math fonts <2025-04-03>} % preloaded in format
```

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

- define `\_unimathfont` as ⟨*Unicode-math font*⟩,
- redefine `\normalmath` and `\boldmath` macros to their Unicode counterparts,
- save the ⟨*optional-factor*⟩ as scaling factor, see also `\_mfactor`,
- load the `\_unimathfont` by `\normalmath`,
- print information about the loaded font on the terminal,
- redefine all encoding dependent setting by `\input unimath-codes.opm`,
- protect new loading by setting `\_ifmathloading` to false.

`\noloadmath` disallows Unicode-math loading by `\_mathloadingfalse`.
`\doloadmath` allows Unicode-math loading by `\_mathloadingtrue`.

unimath-macros.opm

```
\_newifi \_ifmathloading   \_mathloadingtrue

\_def\_noloadmath{\_mathloadingfalse}
\_def\_doloadmath{\_mathloadingtrue}

\_def\_loadmath#1#{\_loadmathA{#1}}
\_def\_loadmathA#1#2{%
   \_ifmathloading
   \_initunifonts
   \_isfont{#2}\_iffalse
      \_opwarning{Math font "#2" not found, skipped...}%
   \_else
      \_sdef{_mfactor:1}{#1}\_def\_unimathfont{#2}%
      \_let\_normalmath = \_normalunimath  \_let\_boldmath = \_boldunimath
      \_normalmath
      \_wterm {MATH-FONT: "#2" -- unicode math prepared.}%
      \_ifx\_ncharrmA\_undefined \_opinput {unimath-codes.opm}\_fi
      \_mathloadingfalse
   \_fi\_fi}

\_public \loadmath \noloadmath \doloadmath ;
```

`\loadboldmath` `{`⟨*bold-font*⟩`} \to {`⟨*normal-font*⟩`}` defines `\_bmath:`⟨*normal-font*⟩ as ⟨*bold-font*⟩. It is used when `\boldmath` macro is run. When no `\_bmath:`⟨*normal-font*⟩ is defined then the `\boldmath` macro use "fake bold" generated by `embolden` LuaTeX font feature.

unimath-macros.opm

```
\_def\_loadboldmath#1#2\to #3{%
   \_isfont{#1}\_iffalse
      \_opwarning{Bold-Math font "#1" not found, skipped...}%
   \_else \_sdef{_bmath:#1}{#3}%
       \_wterm {BOLD-MATH-FONT: "#1" with "#3".}%
   \_fi}

\_public \loadboldmath ;
```

The Unicode version of the `\normalmath` and `\boldmath` macros are defined here as `\_normalunimath` and `\_boldunimath` macros. The base font `\_unimathfont` is loaded to family 1 with `script=math` feature. It is generally used for all variables, digits and math symbols. The same font is loaded to family 2 with `script=latn`. It can be used for sequences of letters where we don't want to insert italic correction between them. The default `\rm` and `\it` macros run `\_rmletters` and `\_itletters` and they use family 2, so these selectors prohibit italic corrections between letters.
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. The final macro `\_setunimathdimens` does similar work as `\_setmathdimens`.

unimath-macros.opm

```
\_def\_normalunimath{%
    \_setmathfamily 0 \_tenrm                         % font for non-math objects in math mode
    \_loadumathfamily 1 {\_unimathfont}{script=math;} % Base math font
    \_loadumathfamily 2 {\_unimathfont}{script=latn;} % Base font, no italic corr. between letters
    \_loadmathfamily  4 rsfs                          % script
    \_setunimathdimens
}%
\_def\_boldunimath{%
    \_setmathfamily 0 \_tenbf                         % font for non-math objects in math mode
    \_ifcsname _bmath:\_unimathfont \_endcsname
       \_loadumathfamily 1 {\_cs{_bmath:\_unimathfont}}{script=math;} % Base real bold font
       \_loadumathfamily 2 {\_cs{_bmath:\_unimathfont}}{script=latn;} % Base bold, no italcorr.
    \_else
       \_loadumathfamily 1 {\_unimathfont}{script=math;embolden=1.7;} % Base faked bold
       \_loadumathfamily 2 {\_unimathfont}{script=latn;embolden=1.7;} % Base faked, no italcor.
    \_fi
    \_loadmathfamily  4 rsfs                          % script
    \_setunimathdimens
}%
\_def\_setunimathdimens{% PlainTeX sets these dimens for 10pt size only:
  \_delimitershortfall=0.5\_fontdimen6\_textfont1
  \_nulldelimiterspace=0.12\_fontdimen6\_textfont1
  \_setmathparam\_Umathspaceafterscript \_scriptspacefactor
  \_setbox0=\_hbox{\_everymath{}$\_fam1\_displaystyle{0\_atop0}$}%
  \_Umathfractiondelsize\_displaystyle = \_dimexpr(\_ht0-\_Umathaxis\_displaystyle)*2\_relax
  \_setbox0=\_box\_voidbox
}
```

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

```
\loadmath{[xitsmath-regular]}
\def\_boldmath{%
   \_loadumathfamily 1 {[xitsmath-bold]}{} % Base font
   \_loadmathfamily 4  rsfs % script
   \_loadumathfamily 5 {[xitsmath-regular]}{}
   \_def\|{\_Udelimiter 0 5 "02016 }%      % norm delimiter from family 5
   \_setmathdimens
}
```

`\_loadumathfamily` ⟨*number*⟩ `{`⟨*font*⟩`}{`⟨*font features*⟩`}` loads the given Unicode-math fonts in three sizes using single ⟨*font*⟩ with different `mathsize=1,2,3` font features. The math font family is set with given ⟨*number*⟩. The ⟨*font features*⟩ are added to the default `\_mfontfeatures` and to the size-dependent features `ssty=1` if script size is asked or `ssty=2` if scriptscriptsize is asked.
`\_mparams` can insert additional font features depending on the current `\_mfam`.
The `\_mfactor` ⟨*family*⟩⟨*space*⟩ sets scaling factor, see section 2.14 for more information.
The `\_textmff`, `\_scriptmff` and `\_sscriptmff` are font features for text, script and sscript sizes respectively. They are locally re-defined in `\mathbox` macro.

unimath-macros.opm

```
\_def\_umathname#1#2{"#1:\_mfontfeatures#2"}
\_def\_mfontfeatures{mode=base;}

\_def\_loadumathfamily{\_afterassignment\_loadumathfamilyA \_chardef\_mfam}
\_def\_loadumathfamilyA #1#2 {\_mfactor
  \_font\_mF \_umathname{#1}{\_textmff   \_mparams #2} at\_sizemtext \_textfont         \_mfam=\_mF
  \_font\_mF \_umathname{#1}{\_scriptmff \_mparams #2} at\_sizemtext \_scriptfont       \_mfam=\_mF
  \_font\_mF \_umathname{#1}{\_sscriptmff\_mparams #2} at\_sizemtext \_scriptscriptfont \_mfam=\_mF
}
\_def\_textmff   {ssty=0;mathsize=1;}
\_def\_scriptmff {ssty=1;mathsize=2;}
\_def\_sscriptmff{ssty=2;mathsize=3;}
\_def\_mparams{}
```

`\addUmathfont` ⟨*fam*⟩ `{[`⟨*normal-font*⟩`]}{`⟨*ffeatures*⟩`}` `{[`⟨*bold-font*⟩`]}{`⟨*ffeatures*⟩`}` `{`⟨*factor*⟩`}` allocates new ⟨*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).

unimath-macros.opm

```
\_def\_addUmathfont #1#2#3#4#5#6{% #1: fam (will be set), #2#3: normal font, #4#5: bold font
   \_ifx\_ncharrmA\_undefined \_errmessage{basic Unicode math font must be loaded first}%
   \_else \_isfont{#2}\_iffalse \_opwarning{font #2 is unavailable}%
   \_else
      \_newfam#1\_relax
      \_sdef{_mfactor:\_the\_numexpr#1\_relax}{#6}%
      \_global\_aheadto\_normalmath{\_loadumathfamily #1{#2}{\_addUmathdefault #3} }%
      \_ifx\_relax#4\_relax
         \_global\_aheadto\_boldmath{\_loadumathfamily #1{#2}{\_addUmathdefault embolden=1.7;#3} }%
      \_else
         \_global\_aheadto\_boldmath{\_loadumathfamily #1{#4}{\_addUmathdefault #5} }%
      \_fi
      \_normalmath
      \_wterm{add-MATH-FONT: #1=\_the#1, "#2", \_ifx"#4"\_else bold: "#4"\_fi}%
   \_fi \_fi
}
\_def\_addUmathdefault {script=math;} % default font features when \_addUmathfont is used
```

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 ⟨*class*⟩, ⟨*family*⟩, ⟨*slot-number*⟩. Sometimes, we may want to get these quantities from the given Unicode math character by our macros. It is possible by `\thematchcodeclass`⟨*math-char*⟩, `\thematchcodefam`⟨*math-char*⟩ and `\thematchcodechar`⟨*math-char*⟩ macros. The parameter ⟨*math-char*⟩ is a math character or it is a macro like `\doteq`, `\times`. Moreower, `\thedelcodefam`⟨*math-char*⟩ and `\thedelcodechar`⟨*math-char*⟩ return delcode quaitities of given math character. All these commands use the common Lua code defined in the `\_getmathcode`⟨*code*⟩`{`⟨*math or del*⟩`}`⟨*character number*⟩ macro.

unimath-macros.opm

```
\_def\_getmathcode#1#2{\_directlua{tex.print(tex.get#2code(token.scan_int())[#1])}}
\_def\_thematchcodeclass #1{\_getmathcode 1{math}\_ea`#1 }
\_def\_thematchcodefam   #1{\_getmathcode 2{math}\_ea`#1 }
\_def\_thematchcodechar  #1{\_getmathcode 3{math}\_ea`#1 }
\_def\_thedelcodefam     #1{\_getmathcode 1{del}\_ea`#1 }
\_def\_thedelcodechar    #1{\_getmathcode 2{del}\_ea`#1 }

\_public \thematchcodeclass \thematchcodefam \thematchcodechar \thedelcodefam \thedelcodechar ;
```

`\mathchars` ⟨*fam*⟩ `{`⟨*list of sequences*⟩`}` saves ⟨*fam*⟩ to `\_mafam` and runs for each sequence from the ⟨*list of sequences*⟩ the relevant code settings using `\Umathcode` primitive. In case of `\int`-like operators the ⟨*math 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*⟩

unimath-macros.opm

```
\_def\_mathchars {\_afterassignment\_mathcharsA \_chardef\_mafam=}
\_def\_mathcharsA #1{\_foreach #1\_do{%
   \_chardef\_tmp=\_thematchcodeclass##1\_relax
   \_ifnum\_tmp=8 % \int, \iint, \oint, etc.
       \_ea\_Umathchardef \_csname _int:##1\_endcsname =1 \_mafam \_ea`##1
   \_else
       \_Umathcode \_ea`##1=\_tmp \_mafam \_thematchcodechar##1
   \_fi
}}
```

`\mathcodes` ⟨*fam*⟩ `{`⟨*list of pairs*⟩`}` sets mathcodes of given characters with explicit ⟨*class*⟩es. Each pair can be ⟨*class*⟩`{`⟨*list of chars*⟩`}` and ⟨*list of chars*⟩ can include `\Urange` ⟨*from*⟩`-`⟨*to*⟩. This is reason why we apply `\expanded` to the ⟨*list of chars*⟩ before reading it by `\foreach`: the `\Urange` is expandable and expands to the relevant list of characters.

unimath-macros.opm

```
\_def\_mathcodes{\_afterassignment\_mathcodesA\_chardef\_mafam=}
\_def\_mathcodesA#1{%
   \_foreach #1\_do ##1##2{%
      \_ea\_foreach\_expanded{##2}\_do{\_Umathcode `####1=##1\_mafam \_ea`####1}%
   }%
}
\_def\_Urange #1-#2{\_fornum \_ea`#1..\_ea`#2\_do{\_Uchar##1 }}

\_public \addUmathfont \mathchars \mathcodes \Urange ;
```

2.16.2 Macros and codes set when `\loadmath` is processed firstly

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

unimath-codes.opm

```
\_codedecl \_ncharrmA {Uni math codes <2025-02-09>} % preloaded on demand by \loadmath
```

Unicode 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.

`\_umathrange` `{`⟨*from*-⟩`-`⟨*to*⟩`}`⟨*class*⟩⟨*family*⟩`\`⟨*first*⟩ sets `\Umathcode`s of the characters in the interval ⟨*from*⟩`-`⟨*to*⟩ to `\`⟨*first*⟩, `\`⟨*first*⟩`+1`, `\`⟨*first*⟩`+2` etc., but `\_umathcharholes` are skipped (`\_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 ⟨*from*⟩`-`⟨*to*⟩ clause includes characters like `A-Z`. Note that the `\_umathrange` sets the `\_classfam` macro as ⟨*class*⟩ ⟨*family*⟩ for later use.

unimath-codes.opm

```
\_newcount\_umathnumA  \_newcount\_umathnumB

\_def\_umathholes[#1]#2{\_ifx^#1^\_else
   \_ea\_chardef \_csname _mh:#1\_endcsname=#2 \_ea\_umathholes\_fi}
\_umathholes % holes in math alphabets:
   [119893]{"210E}[119965]{"212C}[119968]{"2130}[119969]{"2131}%
   [119971]{"210B}[119972]{"2110}[119975]{"2112}[119976]{"2133}[119981]{"211B}%
   [119994]{"212F}[119996]{"210A}[120004]{"2134}%
   [120070]{"212D}[120075]{"210C}[120076]{"2111}[120085]{"211C}[120093]{"2128}%
   [120122]{"2102}[120127]{"210D}[120133]{"2115}[120135]{"2119}%
   [120136]{"211A}[120137]{"211D}[120145]{"2124}[]{}%
\_let\_umathholes=\_undefined

\_def\_umathrange#1#2#3#4{\_umathnumB=#4\_def\_classfam{#2 #3 }\_umathrangeA#1}
\_def\_umathrangeA#1-#2{\_umathnumA=`#1\_relax
   \_loop
      \_Umathcode \_umathnumA = \_classfam \_trycs{_mh:\_the\_umathnumB}\_umathnumB
      \_ifnum\_umathnumA<`#2\_relax
         \_advance\_umathnumA by1 \_advance\_umathnumB by1
   \_repeat
}
```

A few math characters have very specific Unicode and must be handled individually. We can run `\_umathrangespec`⟨*list of characters*⟩`\relax` just after `\_umathrange`. The `\_umathnumB` must be set to the first destination code. The `\_umathrangespec` applies to each character from the ⟨*list of characters*⟩ this: `\Umathcode`⟨*char-code*⟩`=\_classfam\_umathnumB` and increments `\_umathnumB`. If `\_umathnumB=0` then it applies `\Umathcode`⟨*char-code*⟩`=\_classfam` ⟨*char-code*⟩. The `\_classfam` and `\_umathnumB` were typically set by previous call of the `\_umathrange` macro.

unimath-codes.opm

```
\_def\_umathrangespec#1{\_ifx#1\_relax \_else
   \_Umathcode `#1=\_classfam \_ifnum\_umathnumB=0 `#1 \_else \_umathnumB\_fi
   \_unless\_ifnum\_umathnumB=0 \_advance\_umathnumB by1 \_fi
   \_ea\_umathrangespec \_fi
}
```

The math alphabets are set by `\_rmvariables`, `\_rmletters`, `\_bfvariables`, `\_bfletters`, `\_itvariables`, `\_itletters`, `\_bivariables`, `\_biletters`, `\_calvariables`, `\_bcalvariables`, `\_frakvariables`, `\_bfrakvariables`, `\_bbvariables`, `\_sansvariables`, `\_bsansvariables`, `\_isansvariables`, `\_bisansvariables`, `\_ttvariables`,

`\_itgreek`, `\_rmgreek`, `\_bfgreek`, `\_bigreek`, `\_bsansgreek`, `\_bisansgreek`, `\_itGreek`, `\_rmGreek`, `\_bfGreek`, `\_biGreek`, `\_bsansGreek`, `\_bisansGreek`, `\_rmdigits`, `\_bfdigits`, `\_bbdigits`, `\_sansdigits`, `\_bsansdigits`, `\_ttdigits`. They are declared using the `\_umathrange{`⟨*range*⟩`}`⟨*class*⟩⟨*family*⟩⟨*starting-code*⟩ macro.

unimath-codes.opm

```
\_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
\_chardef\_ncharsiA="1D608   \_chardef\_ncharsia="1D622
\_chardef\_ncharsxA="1D63C   \_chardef\_ncharsxa="1D656
\_chardef\_nchalttA="1D670   \_chardef\_nchartta="1D68A

\_protected\_def\_rmvariables     {\_umathrange{A-Z}71\_ncharrmA \_umathrange{a-z}71\_ncharrma}
\_protected\_def\_rmletters       {\_umathrange{A-Z}72\_ncharrmA \_umathrange{a-z}72\_ncharrma}
\_protected\_def\_bfvariables     {\_umathrange{A-Z}71\_ncharbfA \_umathrange{a-z}71\_ncharbfa}
\_protected\_def\_bfletters       {\_umathrange{A-Z}72\_ncharbfA \_umathrange{a-z}72\_ncharbfa}
\_protected\_def\_itvariables     {\_umathrange{A-Z}71\_ncharitA \_umathrange{a-z}71\_ncharita}
\_protected\_def\_itletters       {\_umathrange{A-Z}72\_ncharitA \_umathrange{a-z}72\_ncharita}
\_protected\_def\_bivariables     {\_umathrange{A-Z}71\_ncharbiA \_umathrange{a-z}71\_ncharbia}
\_protected\_def\_biletters       {\_umathrange{A-Z}72\_ncharbiA \_umathrange{a-z}72\_ncharbia}
\_protected\_def\_calvariables    {\_umathrange{A-Z}71\_ncharclA \_umathrange{a-z}71\_ncharcla}
\_protected\_def\_bcalvariables   {\_umathrange{A-Z}71\_ncharbcA \_umathrange{a-z}71\_ncharbca}
\_protected\_def\_frakvariables   {\_umathrange{A-Z}71\_ncharfrA \_umathrange{a-z}71\_ncharfra}
\_protected\_def\_bfrakvariables  {\_umathrange{A-Z}71\_ncharbrA \_umathrange{a-z}71\_ncharbra}
\_protected\_def\_bbvariables     {\_umathrange{A-Z}71\_ncharbbA \_umathrange{a-z}71\_ncharbba}
\_protected\_def\_sansvariables   {\_umathrange{A-Z}71\_ncharsnA \_umathrange{a-z}71\_ncharsna}
\_protected\_def\_bsansvariables  {\_umathrange{A-Z}71\_ncharbsA \_umathrange{a-z}71\_ncharbsa}
\_protected\_def\_isansvariables  {\_umathrange{A-Z}71\_ncharsiA \_umathrange{a-z}71\_ncharsia}
\_protected\_def\_bisansvariables {\_umathrange{A-Z}71\_ncharsxA \_umathrange{a-z}71\_ncharsxa}
\_protected\_def\_ttvariables     {\_umathrange{A-Z}71\_nchalttA \_umathrange{a-z}71\_nchartta}

\_chardef\_greekrmA="0391   \_chardef\_greekrma="03B1
\_chardef\_greekbfA="1D6A8  \_chardef\_greekbfa="1D6C2
\_chardef\_greekitA="1D6E2  \_chardef\_greekita="1D6FC
\_chardef\_greekbiA="1D71C  \_chardef\_greekbia="1D736
\_chardef\_greeksnA="1D756  \_chardef\_greeksna="1D770
\_chardef\_greeksiA="1D790  \_chardef\_greeksia="1D7AA

\_protected\_def\_itgreek    {\_umathrangegreek71\_greekita}
\_protected\_def\_rmgreek    {\_umathrangegreek71\_greekrma}
\_protected\_def\_bfgreek    {\_umathrangegreek71\_greekbfa}
\_protected\_def\_bigreek    {\_umathrangegreek71\_greekbia}
\_protected\_def\_bsansgreek {\_umathrangegreek71\_greeksna}
\_protected\_def\_bisansgreek{\_umathrangegreek71\_greeksia}
\_protected\_def\_itGreek    {\_umathrangeGREEK71\_greekitA}
\_protected\_def\_rmGreek    {\_umathrangeGREEK71\_greekrmA}
\_protected\_def\_bfGreek    {\_umathrangeGREEK71\_greekbfA}
\_protected\_def\_biGreek    {\_umathrangeGREEK71\_greekbiA}
\_protected\_def\_bsansGreek {\_umathrangeGREEK71\_greeksnA}
\_protected\_def\_bisansGreek{\_umathrangeGREEK71\_greeksiA}

\_chardef\_digitrm0=`0
\_chardef\_digitbf0="1D7CE
\_chardef\_digitbb0="1D7D8
\_chardef\_digitsn0="1D7E2
\_chardef\_digitbs0="1D7EC
\_chardef\_digittt0="1D7F6

\_protected\_def\_rmdigits    {\_umathrange{0-9}71\_digitrm0}
\_protected\_def\_bfdigits    {\_umathrange{0-9}71\_digitbf0}
\_protected\_def\_bbdigits    {\_umathrange{0-9}71\_digitbb0}
```

```
\_protected\_def\_sansdigits  {\_umathrange{0-9}71\_digitsn0}
\_protected\_def\_bsansdigits {\_umathrange{0-9}71\_digitbs0}
\_protected\_def\_ttdigits    {\_umathrange{0-9}71\_digittt0}
```

The control sequences for `\alpha`, `\beta`, etc. are redefined here. The `\alpha` will expand to the character with Unicode `"03B1`, this is a normal character α. 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`.

unimath-codes.opm

```
\_def\_greekdef#1{\_ifx#1\_relax
   \_else
      \_edef#1{\_Uchar\_umathnumB}%
      \_advance\_umathnumB by 1
      \_ea\_greekdef \_fi
}
\_umathnumB="0391
\_greekdef \Alpha \Beta \Gamma \Delta \Epsilon \Zeta \Eta \Theta \Iota \Kappa
   \Lambda \Mu \Nu \Xi \Omicron \Pi \Rho \varTheta \Sigma \Tau \Upsilon \Phi
   \Chi \Psi \Omega \_relax

\_umathnumB="03B1
\_greekdef \alpha \beta \gamma \delta \varepsilon \zeta \eta \theta \iota \kappa
   \lambda \mu \nu \xi \omicron \pi \rho \varsigma \sigma \tau \upsilon
   \varphi \chi \psi \omega \_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.

unimath-codes.opm

```
\_def\epsilon{^^^^03f5} \_def\vartheta{^^^^03d1} \_def\varkappa{^^^^03f0}
\_def\phi{^^^^03d5}     \_def\varrho{^^^^03f1}   \_def\varpi{^^^^03d6}
\_def \nabla{^^^^2207}

\_def\_umathrangeGREEK#1#2#3{\_umathrange{^^^^0391-^^^^03a9}#1#2#3% \Alpha-\Omega
   \_resetnabla  % you can do \let\_resetnabla=\relax if you don't want to change \nabla shape
}
\_def\_resetnabla {%
   \_ifnum\_umathnumB<950 \_umathnumB=0 \_else \_advance\_umathnumB by1 \_fi
   \_umathrangespec ^^^^2207\_relax % \nabla
}
\_def\_umathrangegreek#1#2#3{%
   \_umathrange{^^^^03b1-^^^^03c9}#1#2#3% \alpha-\omega
   \_ifnum#3=\_greekrma \_umathnumB=0 \_else \_advance\_umathnumB by2 \_fi
   \_umathrangespec ^^^^03f5^^^^03d1^^^^03f0^^^^03d5^^^^03f1^^^^03d6\_relax % \epsilon-\varpi
}
```

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

```
\_protected\_def\_mabf {\_inmath{\_bfvariables\_bfgreek\_bfGreek\_bfdigits}}
\_protected\_def\_mabi {\_inmath{\_bivariables\_bigreek\_bfGreek\_bfdigits}}
```

`\_inmath` `{`⟨*cmds*⟩`}` applies ⟨*cmds*⟩ only in math mode.

unimath-codes.opm

```
\_protected\_def\_inmath#1{\_relax \_ifmmode#1\_fi} % to keep off \loop processing in text mode

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

\_protected\_def\_marm {\_inmath{\_rmletters \_rmdigits}}
\_protected\_def\_mait {\_inmath{\_itletters \_itGreek}}
\_protected\_def\_mabf {\_inmath{\_bsansvariables \_bsansgreek \_bsansGreek \_bsansdigits}}
\_protected\_def\_mabi {\_inmath{\_bisansvariables \_bisansgreek \_bsansGreek \_bsansdigits}}
\_protected\_def\_matt {\_inmath{\_ttvariables \_ttdigits}}
```

```
\_protected\_def\_bbchar  {\_bbvariables \_bbdigits}
\_protected\_def\_cal     {\_calvariables}
\_protected\_def\_frak    {\_frakvariables}
\_protected\_def\_misans  {\_isansvariables \_sansdigits}
\_protected\_def\_mbisans {\_bisansvariables \_bisansgreek \_bsansGreek \_bsansdigits}
\_protected\_def\_script  {\_rmvariables \_fam4 }
\_protected\_def\_mit     {\_itvariables \_rmdigits \_itgreek \_rmGreek }

\_public \bbchar \cal \frak \misans \mbisans \script \mit ;
```

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

`mathclass.opm`

```
002E;P
002F;B
0030..0039;N
003A;P
003B;P
003C;R
003D;R
003E;R
003F;P
0040;N
0041..005A;A
005B;O
005C;B
005D;C
005E;N
005F;N
```

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

`unimath-codes.opm`

```
\_begingroup  % \input mathclass.opm  (which is a copy of MathClass.txt):
   \_long\_def\_p#1;#2 {\_ifx^#2^\_else
      \_edef\_tmp{\_csname _c:#2\_endcsname}\_if\_relax\_tmp\_else \_pA#1....\_end#2\_fi
      \_ea\_p \_fi }
   \_def\_pA#1..#2..#3\_end#4{%
      \_ifx\_relax#2\_relax \_pset{"#1}{#4}\_else \_fornum "#1.."#2\_do{\_pset{##1}{#4}}\_fi
   }
   \_sdef{_c:L}{1}\_sdef{_c:B}{2}\_sdef{_c:V}{2}\_sdef{_c:R}{3}\_sdef{_c:N}{0}\_sdef{_c:U}{0}
   \_sdef{_c:F}{0}\_sdef{_c:O}{4}\_sdef{_c:C}{5}\_sdef{_c:P}{6}\_sdef{_c:A}{7}
   \_def\_pset#1#2{\_Umathcode#1=\_tmp\_space 1 #1\_relax
      \_if#2O\_Udelcode#1=1 #1\_relax\_fi
      \_if#2C\_Udelcode#1=1 #1\_relax\_fi
      \_if#2F\_Udelcode#1=1 #1\_relax\_fi
   }
   \_catcode`#=14 \_everyeof={;{} } \_def\par{}
   \_globaldefs=1 \_ea \_p \_input mathclass.opm
\_endgroup
```

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

`unimath-table.opm`

```
\UnicodeMathSymbol{"000B1}{\pm                       }{\mathbin}{plus-or-minus sign}%
\UnicodeMathSymbol{"000B6}{\mathparagraph            }{\mathord}{paragraph symbol}%
\UnicodeMathSymbol{"000B7}{\cdotp                    }{\mathbin}{/centerdot b: middle dot}%
\UnicodeMathSymbol{"000D7}{\times                    }{\mathbin}{multiply sign}%
\UnicodeMathSymbol{"000F0}{\matheth                  }{\mathalpha}{eth}%
\UnicodeMathSymbol{"000F7}{\div                      }{\mathbin}{divide sign}%
```

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

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

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

`unimath-codes.opm`

```
\_begingroup  % \input unimath-table.opm (it is a copy of unicode-math-table.tex):
   \_umathnumB="F800 % pointer to the Private User Area
   \_def\UnicodeMathSymbol #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.opm
      \_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.opm
\_endgroup
```

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

`unimath-codes.opm`

```
\_def\_intwithnolimits#1{\_ifx#1\_relax \_else
   \_ea\_Umathcharnumdef\_csname _int:#1\_endcsname=\_Umathcodenum\_ea`#1 %
   \_ea\_def \_csname\_csstring#1op\_ea\_endcsname\_ea{\_csname _int:#1\_endcsname}%
   \_bgroup \_lccode`=\_ea`#1 \_lowercase{\_egroup
      \_ea\_def\_ea~\_ea{\_csname _int:#1\_endcsname\_nolimits}\_mathcode`~="8000 }%
   \_ea \_intwithnolimits \_fi
}
\_intwithnolimits \int \iint \iiint \oint \oiint \oiiint
   \intclockwise \varointclockwise \ointctrclockwise \sumint \iiiint \intbar \intBar \fint
   \pointint \sqint \intlarhk \intx \intcap \intcup \upint \lowint \_relax
```

Many special characters must be declared with care...

`unimath-codes.opm`

```
\_global\_Udelcode`<=1 "027E8 % these characters have different meaning
\_global\_Udelcode`>=1 "027E9 % as normal and as delimiter

\_mit % default math alphabets setting

% hyphen character is transformed to minus:
\_Umathcode `- = 2 1 "2212

% mathclass defines : as Punct, plain.tex as Rel, we keep mathclass,
% i.e. there is difference from plain.tex, you can use $f:A\to B$.

% mathclas defines ! as Ord, plain.tex as Close
\_Umathcode `! = 5 1 `!  % keep plain.tex declaration
% mathclas defines ? as Punct, plain.tex as Close
\_Umathcode `? = 5 1 `?  % keep plain.tex declaration

\_Umathcode `* = 2 1 "02217  % equivalent to \ast, like in plain TeX

\_Umathcode "03A2 = 7 1 "03F4 % \varTheta

\_Umathcode `© = 0 1 `© % usage $\copyright$ can be seen in old documents

\_protected\_def \_sqrt       {\_Uradical 1 "0221A }
\_protected\_def \_cuberoot   {\_Uradical 1 "0221B }
\_protected\_def \_fourthroot {\_Uradical 1 "0221C }

\_public \sqrt \cuberoot \fourthroot ;

\_protected\_def \_overbrace    #1{\_mathop {\_Umathaccent  7 1 "023DE{#1}}\_limits}
\_protected\_def \_underbrace   #1{\_mathop {\_Umathaccent bottom 7 1 "023DF{#1}}\_limits}
\_protected\_def \_overparen    #1{\_mathop {\_Umathaccent  7 1 "023DC{#1}}\_limits}
\_protected\_def \_underparen   #1{\_mathop {\_Umathaccent bottom 7 1 "023DD{#1}}\_limits}
```

```
\_protected\_def \_overbracket  #1{\_mathop {\_Umathaccent  7 1 "023B4{#1}}\_limits}
\_protected\_def \_underbracket #1{\_mathop {\_Umathaccent bottom 7 1 "023B5{#1}}\_limits}

\_public \overbrace \underbrace \overparen \underparen \overbracket \underbracket ;

\_protected\_def \widehat            {\_Umathaccent 7 1 "00302 }
\_protected\_def \widetilde          {\_Umathaccent 7 1 "00303 }
\_protected\_def \overleftharpoon    {\_Umathaccent 7 1 "020D0 }
\_protected\_def \overrightharpoon   {\_Umathaccent 7 1 "020D1 }
\_protected\_def \overleftarrow      {\_Umathaccent 7 1 "020D6 }
\_protected\_def \overrightarrow     {\_Umathaccent 7 1 "020D7 }
\_protected\_def \overleftrightarrow {\_Umathaccent 7 1 "020E1 }

\_protected\_def \wideoverbar  {\_Umathaccent 7 1 "00305 }
\_protected\_def \widebreve    {\_Umathaccent 7 1 "00306 }
\_protected\_def \widecheck    {\_Umathaccent 7 1 "0030C }
\_protected\_def \wideutilde   {\_Umathaccent bottom 7 1 "00330 }
\_protected\_def \mathunderbar {\_Umathaccent bottom 7 1 "00332 }
\_protected\_def \underleftrightarrow   {\_Umathaccent bottom 7 1 "0034D }
\_protected\_def \widebridgeabove       {\_Umathaccent 7 1 "020E9 }
\_protected\_def \underrightharpoondown {\_Umathaccent bottom 7 1 "020EC }
\_protected\_def \underleftharpoondown  {\_Umathaccent bottom 7 1 "020ED }
\_protected\_def \underleftarrow        {\_Umathaccent bottom 7 1 "020EE }
\_protected\_def \underrightarrow       {\_Umathaccent bottom 7 1 "020EF }

\_mathchardef\ldotp="612E
\_let=\Vert
\_mathcode`\_="8000

\_global\_Umathcode    "22EF         = 0 1 "22EF % mathclass says that it is Rel
\_global\_Umathcode    "002E         = 0 1 "002E % mathclass says that dot is Punct

\_global\_Umathcode `/ = 0 1 `/  % mathclass says that / is Bin, Plain TeX says that it is Ord.

% compressed dots in S and SS styles (usable in \matrix when it is in T, S and SS style)
\_protected\_def \vdots {\_relax \_ifnum \_mathstyle>3 \_unicodevdots \_else \_vdots \_fi}
\_protected\_def \ddots {\_relax \_ifnum \_mathstyle>3 \_unicodeddots \_else \_ddots \_fi}
\_protected\_def \adots {\_relax \_ifnum \_mathstyle>3 \_unicodeadots \_else \_adots \_fi}

% Unicode superscripts (²) and subscripts as simple macros with \mathcode"8000
\_bgroup
   \_def\_tmp#1#2{\_global\_mathcode#1="8000 \_lccode`~=#1 \_lowercase{\_gdef~}{#2}}
   \_fornum 0..1 \_do {\_tmp{"207#1}{{^#1}}}
   \_tmp{"B2}{{^2}}\_tmp{"B3}{{^3}}
   \_fornum 4..9 \_do {\_tmp{"207#1}{{^#1}}}
   \_fornum 0..9 \_do {\_tmp{"208#1}{{_#1}}}
\_egroup
```

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

unimath-codes.opm

```
\_let \setminus=\smallsetminus
\_let \diamond=\smwhtdiamond
\_let \colon=\mathcolon
\_let \bullet=\smblkcircle
\_let \circ=\vysmwhtcircle
\_let \bigcirc=\mdlgwhtcircle
\_let \to=\rightarrow
\_let \le=\leq
\_let \ge=\geq
\_let \neq=\ne
\_protected\_def \triangle {\mathord{\bigtriangleup}}
\_let \emptyset=\varnothing
\_let \hbar=\hslash
\_let \land=\wedge
\_let \lor=\vee
\_let \owns=\ni
\_let \gets=\leftarrow
\_let \mathring=\ocirc
```

```
\_let \lnot=\neg
\_let \longdivisionsign=\longdivision
\_let \backepsilon=\upbackepsilon
\_let \eth=\matheth
\_let \dbkarow=\dbkarrow
\_let \drbkarow=\drbkarrow
\_let \hksearow=\hksearrow
\_let \hkswarow=\hkswarrow
\_let \square=\mdlgwhtsquare
\_let \blacksquare=\mdlgblksquare

\_let \upalpha=\mupalpha
\_let \upbeta=\mupbeta
\_let \upgamma=\mupgamma
\_let \updelta=\mupdelta
\_let \upepsilon=\mupvarepsilon
\_let \upvarepsilon=\mupvarepsilon
\_let \upzeta=\mupzeta
\_let \upeta=\mupeta
\_let \uptheta=\muptheta
\_let \upiota=\mupiota
\_let \upkappa=\mupkappa
\_let \uplambda=\muplambda
\_let \upmu=\mupmu
\_let \upnu=\mupnu
\_let \upxi=\mupxi
\_let \upomicron=\mupomicron
\_let \uppi=\muppi
\_let \uprho=\muprho
\_let \upvarrho=\mupvarrho
\_let \upvarsigma=\mupvarsigma
\_let \upsigma=\mupsigma
\_let \uptau=\muptau
\_let \upupsilon=\mupupsilon
\_let \upvarphi=\mupvarphi
\_let \upchi=\mupchi
\_let \uppsi=\muppsi
\_let \upomega=\mupomega
\_let \upvartheta=\mupvartheta
\_let \upphi=\mupphi
\_let \upvarpi=\mupvarpi
\_let \varTheta=\mupvarTheta
\_let \vardelta=\delta
```

The `\not` macro is redefined here. If the `\_not!`⟨*char*⟩ is defined (by `\_negationof`) then this macro is used. Else centered / is printed over the ⟨*char*⟩.

unimath-codes.opm

```
\_protected\_def\_not#1{%
  \_trycs{_not!\_csstring#1}{\_mathrel\_mathstyles{%
     \_setbox0=\_hbox{\_math$\_currstyle#1$}%
     \_hbox to\_wd0{\_hss$\_currstyle/$\_hss}\_kern-\_wd0 \_box0
}}}
\_def\_negationof #1#2{\_ea\_let \_csname _not!\_csstring#1\_endcsname =#2}

\_negationof =         \neq
\_negationof <         \nless
\_negationof >         \ngtr
\_negationof \gets     \nleftarrow
\_negationof \simeq    \nsime
\_negationof \equal    \ne
\_negationof \le       \nleq
\_negationof \ge       \ngeq
\_negationof \greater  \ngtr
\_negationof \forksnot \forks
\_negationof \in       \notin
\_negationof \mid      \nmid
\_negationof \cong     \ncong
\_negationof \leftarrow \nleftarrow
\_negationof \rightarrow \nrightarrow
```

```
\_negationof \leftrightarrow \nleftrightarrow
\_negationof \Leftarrow \nLeftarrow
\_negationof \Leftrightarrow \nLeftrightarrow
\_negationof \Rightarrow \nRightarrow
\_negationof \exists  \nexists
\_negationof \ni      \nni
\_negationof \parallel \nparallel
\_negationof \sim     \nsim
\_negationof \approx  \napprox
\_negationof \equiv   \nequiv
\_negationof \asymp   \nasymp
\_negationof \lesssim \nlesssim
\_negationof \ngtrsim \ngtrsim
\_negationof \lessgtr \nlessgtr
\_negationof \gtrless \ngtrless
\_negationof \prec    \nprec
\_negationof \succ    \nsucc
\_negationof \subset  \nsubset
\_negationof \supset  \nsupset
\_negationof \subseteq \nsubseteq
\_negationof \supseteq \nsupseteq
\_negationof \vdash   \nvdash
\_negationof \vDash   \nvDash
\_negationof \Vdash  \nVdash
\_negationof \VDash  \nVDash
\_negationof \preccurlyeq \npreccurlyeq
\_negationof \succcurlyeq \nsucccurlyeq
\_negationof \sqsubseteq \nsqsubseteq
\_negationof \sqsupseteq \nsqsupseteq
\_negationof \vartriangleleft \nvartriangleleft
\_negationof \vartriangleright \nvartriangleright
\_negationof \trianglelefteq \ntrianglelefteq
\_negationof \trianglerighteq \ntrianglerighteq
\_negationof \vinfty \nvinfty

\_public \not ;
```

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

unimath-codes.opm

```
\_private
   \ldotp \cdotp \bullet \triangleleft \triangleright \mapstochar \rightarrow
   \prime \lhook \rightarrow \leftarrow \rhook \triangleright \triangleleft
   \rbrace \lbrace \Relbar \Rightarrow \relbar \rightarrow \Leftarrow \mapstochar
   \longrightarrow \Longleftrightarrow \unicodevdots \unicodeddots \unicodeadots ;
```

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.

See `http://tex.stackexchange.com/questions/308749` for technical details about Unicode-math.

2.16.4 Printing all Unicode math slots in used math font

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

Load Unicode math font first (for example by `\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.

print-unimath.opm

```
\_codedecl \_undefined {Printing Unicode-math table \string<2020-06-08>}

\_ifx\_ncharrmA\_undefined \_opwarning{No Unicode math font loaded, printing ignored}
   \_endinput \_fi

\_begingroup
   \_def\UnicodeMathSymbol#1#2#3#4{%
      \_ifnum#1>"10000 \_endinput \_else \_printmathsymbol{#1}{#2}{#3}{#4}\_fi
   }
   \_def\UnicodeMathSymbolA#1#2#3#4{%
      \_ifnum#1>"10000 \_printmathsymbol{#1}{#2}{#3}{#4}\_fi
   }
   \_def\_printmathsymbol#1#2#3#4{%
      \_hbox{\_hbox to2em{$#2{}$\_hss}\_hbox to3em
          {\small\_printop#3\_hss}{\_tt\_string#2\_trycs{_eq:\_string#2}{}}}
   }
   \_def\_eq#1#2{\_sdef{_eq:\_string#2}{=\_string#1}}
   \_eq \diamond\smwhtdiamond \_eq \bullet\smblkcircle \_eq \circ\vysmwhtcircle
   \_eq \bigcirc\mdlgwhtcircle \_eq \to\rightarrow \_eq \le\leq
   \_eq \ge\geq \_eq \neq\ne \_eq \emptyset\varnothing \_eq \hbar\hslash
   \_eq \land\wedge \_eq \lor\vee \_eq \owns\ni \_eq \gets\leftarrow
   \_eq \mathring\ocirc \_eq \lnot\neg \_eq \backepsilon\upbackepsilon
   \_eq \eth\matheth \_eq \dbkarow\dbkarrow \_eq \drbkarow\drbkarrow
   \_eq \hksearow\hksearrow  \_eq \hkswarow\hkswarrow

   \_tracinglostchars=0
   \_fontdef\small{\_setfontsize{at5pt}\_rm}
   \_def\_printop{\_def\mathop{Op}}
   \_def\mathalpha{Alph}\_def\mathord{Ord}\_def\mathbin{Bin}\_def\mathrel{Rel}
   \_def\mathopen{Open}\_def\mathclose{Close}\_def\mathpunct{Punct}\_def\mathfence{Fence}
   \_def\mathaccent{Acc}\_def\mathaccentwide{Accw}\_def\mathbotaccentwide{AccBw}
   \_def\mathbotaccent{AccB}\_def\mathaccentoverlay{AccO}
   \_def\mathover{Over}\_def\mathunder{Under}
   \_typosize[7.5/9]\_normalmath \_everymath={}

   Codes U+00000 \_dotsU+10000
   \_begmulti 3
      \_input unimath-table.opm
   \_endmulti

   \_medskip\_goodbreak
   Codes U+10001 \_dotsU+1EEF1  \_let\UnicodeMathSymbol=\UnicodeMathSymbolA
   \_begmulti 4
      \_input unimath-table.opm
   \_endmulti
\_endgroup
```

2.17 Scaling fonts in document (high-level macros)

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

fonts-opmac.opm

```
\_codedecl \typosize {Font managing macros from OPmac <2022-02-22>} % preloaded in format
```

`\typosize` [⟨*font-size*⟩/⟨*baselineskip*⟩] sets given parameters. It sets text font size by the `\setfontsize` macro and math font sizes by setting internal macros `\_sizemtext`, `\_sizemscript` and `\_sizemsscript`. It uses common concept font sizes: 100 %, 70 % and 50 %. The `\_setmainvalues` sets the parameters as main values when the `\_typosize` is called first.

fonts-opmac.opm

```
\_protected\_def \_typosize [#1/#2]{%
   \_textfontsize{#1}\_mathfontsize{#1}\_setbaselineskip{#2}%
   \_setmainvalues \_ignorespaces
}
\_protected\_def \_textfontsize #1{\_if$#1$\_else \_setfontsize{at#1\_ptunit}\_fi}

\_def \_mathfontsize #1{\_if$#1$\_else
    \_tmpdim=#1\_ptunit
    \_edef\_sizemtext{\_ea\_ignorept \_the\_tmpdim \_ptmunit}%
    \_tmpdim=0.7\_tmpdim
    \_edef\_sizemscript{\_ea\_ignorept \_the\_tmpdim \_ptmunit}%
    \_tmpdim=#1\_ptunit \_tmpdim=0.5\_tmpdim
    \_edef\_sizemsscript{\_ea\_ignorept \_the\_tmpdim \_ptmunit}%
    \_fi
}
\_public \typosize ;
```

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

fonts-opmac.opm

```
\_protected\_def \_typoscale [#1/#2]{%
   \_ifx$#1$\_def\_tmp{[/}\_else
      \_settmpdim{#1}\_optsize
      \_edef\_tmp{[\_ea\_ignorept\_the\_tmpdim/}\_fi
   \_ifx$#2$\_edef\_tmp{\_tmp]}\_else
      \_settmpdim{#2}\_baselineskip
      \_edef\_tmp{\_tmp \_ea\_ignorept\_the\_tmpdim]}\_fi
   \_ea\_typosize\_tmp
}
\_def\_settmpdim#1#2{%
   \_tmpdim=#1pt \_divide\_tmpdim by1000
   \_tmpdim=\_ea\_ignorept \_the#2\_tmpdim
}
\_public \typoscale ;
```

`\_setbaselineskip` `{`⟨*baselineskip*⟩`}` sets new `\baselineskip` and more values of registers which are dependent on the ⟨*baselineskip*⟩ including the `\strutbox`.

fonts-opmac.opm

```
\_def \_setbaselineskip #1{\_if$#1$\_else
   \_tmpdim=#1\_ptunit
   \_baselineskip=\_tmpdim \_relax
   \_bigskipamount=\_tmpdim plus.33333\_tmpdim minus.33333\_tmpdim
   \_medskipamount=.5\_tmpdim plus.16666\_tmpdim minus.16666\_tmpdim
   \_smallskipamount=.25\_tmpdim plus.08333\_tmpdim minus.08333\_tmpdim
   \_normalbaselineskip=\_tmpdim
   \_jot=.25\_tmpdim
   \_maxdepth=.33333\_tmpdim
   \_setbox\_strutbox=\_hbox{\_vrule height.709\_tmpdim depth.291\_tmpdim width0pt}%
   \_fi
}
```

`\_setmainvalues` sets the current font size and `\baselineskip` values to the `\mainfosize` and `\mainbaselineskip` registers and loads fonts at given sizes. It 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 10/12 pt.
`\mfontsrule` gives the rule how math fonts are loaded when `\typosize` or `\typoscale` are used. The value of `\mfontsrule` can be:

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

fonts-opmac.opm

```
\_newskip   \_mainbaselineskip   \_mainbaselineskip=0pt \_relax
\_newdimen  \_mainfosize         \_mainfosize=0pt
\_newcount  \_mfontsrule         \_mfontsrule=1

\_def\_setmainvalues {%
   \_mainbaselineskip=\_baselineskip
   \_mainfosize=\_optsize
   \_topskip=\_mainfosize \_splittopskip=\_topskip
   \_ifmmode \_else \_rm \_fi                % load and initialize \rm variant
   \_ifnum \_mfontsrule>0 \_normalmath \_fi  % load math fonts first
   \_let \_setmainvalues =\_setmainvaluesL
}
\_def\_setmainvaluesL {\_relax \_ifmmode \_else \_rm \_fi % load text font
   \_ifcase \_mfontsrule                                  % load math fonts
   \_or \_ifnum\_currentgrouplevel=0 \_normalmath
        \_else \_everymath={\_setmathfonts}\_everydisplay={\_normalmath}%
               \_let\_runboldmath=\_relax \_fi
   \_or \_normalmath \_fi}
\_def\_scalemain {%
   \_ifdim \_mainfosize=\_zo
       \_mainfosize=10pt  \_mainbaselineskip=12pt
       \_let \_setmainvalues=\_setmainvaluesL
    \_fi
   \_optsize=\_mainfosize  \_baselineskip=\_mainbaselineskip
}
\_public \scalemain \mainfosize \mainbaselineskip \mfontsrule ;
```

Suppose following example: `{\typosize[13/15] Let $M$ be a subset of $R$ and $x\in M$...}` If `\mfontsrule=1` then `\typosize` does not load math fonts immediately but at the first math formula. It is done by `\everymath` register, but the contents of this 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.

`fonts-opmac.opm`

```
\_def\_setmathfonts{$\_normalmath\_everymath{}\_everydisplay{}$}
```

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

`fonts-opmac.opm`

```
\_protected\_def\_thefontsize[#1]{\_if$#1$\_else
     \_tmpdim=#1\_ptunit
     \_newcurrfontsize{at\_tmpdim}%
  \_fi
  \_ignorespaces
}
\_protected\_def\_thefontscale[#1]{\_ifx$#1$\_else
     \_tmpdim=#1pt \_divide\_tmpdim by1000
     \_tmpdim=\_ea\_ea\_ea\_ignorept \_pdffontsize\_font \_tmpdim
     \_newcurrfontsize{at\_tmpdim}%
  \_fi
  \_ignorespaces
}
\_public \thefontsize \thefontscale ;
```

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

`fonts-opmac.opm`

```
\_protected\_def\_em {%
   \_ea\_ifx \_the\_font \_tenit \_additcorr \_rm  \_else
   \_ea\_ifx \_the\_font \_tenbf \_bi\_aftergroup\_afteritcorr\_else
   \_ea\_ifx \_the\_font \_tenbi \_additcorr \_bf  \_else
   \_it \_aftergroup\_afteritcorr\_fi\_fi\_fi
}
\_def\_additcorr{\_ifhmode \_ifdim\_lastskip>\_zo
   \_skip0=\_lastskip \_unskip \_additcorrA \_hskip\_skip0 \_else \_additcorrA \_fi\_fi}
\_def\_additcorrA{\_ifnum\_lastpenalty=\_zo \_italcorr \_else
    \_ea\_unpenalty \_ea\_italcorr \_ea\_penalty \_the\_lastpenalty \_relax \_fi}
\_def\_afteritcorr{\_futurelet\_next\_afteritcorrA}
\_def\_afteritcorrA{\_ifhmode \_ifx\_next.\_else\_ifx\_next,\_else \_italcorr \_fi\_fi\_fi}
\_let\_italcorr=
```

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.

`fonts-opmac.opm`

```
\_protected\_def \_boldify {%
   \_let \_setmainvalues=\_setmainvaluesL
   \_let\it =\_bi \_let\rm =\_bf \_let\_normalmath=\_boldmath \_bf
   \_runboldmath
   \_ifx\_ncharrmA\_undefined \_protected\_addto\rm{\_fam0 }\_protected\_addto\it{\_fam1 }%
   \_else \_protected\_def\rm {\_fmodbf \_fontsel \_marm}%
          \_protected\_def\it {\_fmodbi \_fontsel \_mait}%
   \_fi
}
\_def\_runboldmath{\_boldmath}

\_public \em \boldify ;
```

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.

fonts-opmac.opm

```
\_def \_rmfixed {% used in default \footline
   {\_ifdim\_mainfosize=0pt \_mainfosize=10pt \_fi
    \_fontdef\_tenrm{\_setfontsize{at\mainfosize}\_resetmod\_rm}%
    \_glet\_rmfixed=\_tenrm}% next use will be font switch only
   \_rmfixed
}
\_let \rmfixed = \_tenrm % user can redefine it
```

2.18 Output routine

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

- _begoutput which does:
  - increments \gpageno,
  - prints _Xpage{⟨*gpageno*⟩}{⟨*pageno*⟩} to the .ref file (if \openref is active),
  - calculates \hoffset,
  - sets local meaning of macros used in headlines/footlines (see \regmacro).
- \shipout_completepage, which is \vbox of –
  - backrground box, if \pgbackground is non-empty,
  - headline box by _makeheadline, if the \headline is nonempty,
  - \vbox to\vsize of _pagecontents which cosnists of –
    - _pagedest, the page destination pg:⟨*gpageno*⟩ for hyperlinks is created here,
    - \topins box if non-empty (from \topinserts),
    - \box255 with completed vertical material from main vertical mode,
    - _footnoterule and \footins box if nonempty (from \fnote, \footnote),
    - \pgbottomskip (default is 0 pt).
  - footline box by _makefootline, if the \footline is nonempty
- _endoutput which does:
  - increments \pageno using \advancepageno
  - runs output routine repeatedly if \dosupereject is activated.

output.opm

```
\_codedecl \nopagenumbers {Output routine <2025-05-04>} % preloaded in format
```

_optexoutput is the default output routine. You can create another

output.opm

```
\_output={\_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.

output.opm

```
\_def \_begoutput{\_incr\_gpageno
   \_immediate\_wref\_Xpage{{\_the\_gpageno}{\_folio}}%
   \_setxhsize \_prepoffsets \_the\_regmark}
\_def \_endoutput{\_advancepageno
   {\_globaldefs=1 \_the\_nextpages \_nextpages={}}%
   \_ifnum\_outputpenalty>-20000 \_else\_dosupereject\_fi
}
\_def \_prepoffsets {}
```

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.

output.opm

```
\_def \_optexshipout #1{\_setbox0=#1\_preshipout0\_box0 \_shipout\_box0 }
```

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 beginning of the `\begtt`...`\endtt` environment before `\hsize` value is eventually changed by the user in this environment.

output.opm

```
\_newdimen \_xhsize  \_xhsize=\_hsize
\_def\_setxhsize {\_global\_xhsize=\_hsize \_glet\_setxhsize=\_relax}
```

`\gpageno` counts pages from one in the whole document

output.opm

```
\_newcount\_gpageno
\_public \gpageno ;
```

The `\_completepage` is similar to what plain TEX does in its output routine. New is only `\_backgroundbox`. It is `\vbox` with zero height with its contents (from `\pgbackground`) extended down. It is shifted directly to the left-upper corner of the paper.
The `\_resetattrs` used here means that all newly created texts in output routine (texts used in headline, footline) have default color and no transparency.

output.opm

```
\_def\_completepage{\_vbox{%
     \_resetattrs
     \_istoksempty \_pgbackground
        \_iffalse \_backgroundbox{\_the\_pgbackground}\_nointerlineskip \_fi
     \_makeheadline
     \_vbox to\_vsize {\_boxmaxdepth=\_maxdepth \_pagecontents}% \pagebody in plainTeX
     \_makefootline}%
}
\_def \_backgroundbox #1{\_moveleft\_hoffset\_vbox to\_zo{\_kern-\_voffset #1\_vss}}
```

`\_makeheadline` creates `\vbox to0pt` with its contents (the `\headline`) shifted by `\headlinedist` up.

output.opm

```
\_def\_makeheadline {\_istoksempty \_headline \_iffalse
   \_vbox to\_zo{\_vss \_pdfrunninglinkoff
                 \_baselineskip=\_headlinedist \_lineskiplimit=-\_maxdimen
                 \_hbox to\_xhsize{\_normalbaselines\_the\_headline}\_hbox{}%
                 \_pdfrunninglinkon}\_nointerlineskip
   \_fi
}
```

The `\_makefootline` appends the `\footline` to the page-body box.

output.opm

```
\_def\_makefootline{\_istoksempty \_footline \_iffalse
      \_baselineskip=\_footlinedist
      \_lineskiplimit=-\_maxdimen \_hbox to\_xhsize{\_normalbaselines\_the\_footline}
   \_fi
}
```

The `\_pagecontents` is similar as in plain TEX. The only difference is that the `\_pagedest` is inserted at the top of `\_pagecontents`.
The `\_footnoterule` is defined here.

output.opm

```
\_def\_pagecontents{\_pagedest % destination of the page
  \_ifvoid\_topins \_else \_unvbox\_topins\_fi
  \_dimen0=\_dp255 \_unvbox255 % open up \box255
  \_ifvoid\_footins \_else % footnote info is present
    \_pdfrunninglinkoff \_vskip\_skip\_footins
    \_footnoterule \_unvbox\_footins \_pdfrunninglinkon \_fi
  \_kern-\_dimen0 \_vskip \_pgbottomskip
}
\_def \_pagedest {{\_def\_destheight{25pt}\_dest[pg:\_the\_gpageno]}}
\_def \_footnoterule {\_kern-3pt \_hrule width 2truein \_kern 2.6pt }
```

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

output.opm

```
\_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 ;
```

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

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

The `\footnote` and `\footstrut` is defined as in plain TeX.

output.opm

```
\_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 \footnote \vfootnote \footstrut ;
```

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

output.opm

```
\_newinsert\_topins
\_newifi\_ifupage \_newifi\_ifumid
\_def \_topinsert {\_umidfalse \_upagefalse \_oins}
\_def \_midinsert {\_umidtrue \_oins}
\_def \_pageinsert {\_umidfalse \_upagetrue \_oins}
\_skip\_topins=\_zoskip % no space added when a topinsert is present
\_count\_topins=1000 % magnification factor (1 to 1)
\_dimen\_topins=\_maxdimen % no limit per page
\_def \_oins {\_par \_begingroup\_setbox0=\_vbox\_bgroup\_resetattrs} % start a \_vbox
\_def \_endinsert {\_par\_egroup % finish the \_vbox
  \_ifumid \_dimen0=\_ht0 \_advance\_dimen0 by\_dp0 \_advance\_dimen0 by\_baselineskip
    \_advance\_dimen0 by\_pagetotal \_advance\_dimen0 by-\_pageshrink
    \_ifdim\_dimen0>\_pagegoal \_umidfalse \_upagefalse \_fi \_fi
  \_ifumid \_bigskip \_box0 \_bigbreak
  \_else \_insert \_topins {\_penalty100 % floating insertion
    \_splittopskip=0pt
    \_splitmaxdepth=\_maxdimen \_floatingpenalty=0
    \_ifupage \_dimen0=\_dp0
    \_vbox to\_vsize {\_unvbox0 \_kern-\_dimen0}% depth is zero
    \_else \_box0 \_nobreak \_bigskip \_fi}\_fi\_endgroup}

\_public \topins \topinsert \midinsert \pageinsert \endinsert ;
```

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

output.opm

```
\_def \_draft {\_pgbackground={\_draftbox{\_draftfont DRAFT}}%
   \_fontdef\_draftfont{\_setfontsize{at10pt}\_bf}%
   \_glet\_draftfont=\_draftfont
}
\_def \_draftbox #1{\_setbox0=\_hbox{\_setgreycolor{.8}#1}%
   \_kern.5\_vsize \_kern\_voffset \_kern4.5\_wd0
   \_hbox to0pt{\_kern.5\_xhsize \_kern\_hoffset \_kern-2\_wd0
   \_pdfsave \_pdfrotate{55}\_pdfscale{10}{10}%
   \_hbox to0pt{\_box0\_hss}%
   \_pdfrestore
   \_hss}%
}
\_public \draft ;
```

2.19 Margins

The `\margins` macro is documented in the section 1.2.1.

margins.opm

```
\_codedecl \margins {Macros for margins setting <2023-05-01>} % preloaded in format
```

`\margins`/⟨*pg*⟩ ⟨*fmt*⟩ (⟨*left*⟩,⟨*right*⟩,⟨*top*⟩,⟨*bot*⟩)⟨*unit*⟩ takes its parameters, does calculation and sets `\hoffset`, `\voffset`, `\hsize` and `\vsize` registers. Note that OpTeX sets the page origin at the top left corner of the paper, no at the obscure position 1 in, 1 in. It is much more comfortable for macro writers.

margins.opm

```
\_newdimen\_pgwidth  \_newdimen\_pgheight  \_pgwidth=0pt
\_newdimen\_shiftoffset

\_def\_margins/#1 #2 (#3,#4,#5,#6)#7 {\_def\_tmp{#7}%
   \_ifx\_tmp\_empty
      \_opwarning{\_string\_margins: missing unit, mm inserted}\_def\_tmp{mm}\_fi
   \_setpagedimens #2 % setting \_pgwidth, \_pgheight
   \_ifdim\_pgwidth=0pt \_else
      \_hoffset=0pt \_voffset=0pt
      \_if$#3$\_if$#4$\_hoffset =\_dimexpr (\_pgwidth -\_hsize)/2 \_relax
              \_else  \_hoffset =\_dimexpr \_pgwidth -\_hsize - #4\_tmp \_relax % only right margin
              \_fi
      \_else  \_if$#4$\_hoffset = #3\_tmp \_relax  % only left margin
              \_else  \_hsize =\_dimexpr \_pgwidth - #3\_tmp - #4\_tmp \_relax % left+right margin
                      \_hoffset = #3\_tmp \_relax
                      \_xhsize =\_hsize \_setxhsize % \_xhsize used by \output routine
      \_fi\_fi
      \_if$#5$\_if$#6$\_voffset =\_dimexpr (\_pgheight -\_vsize)/2 \_relax
              \_else  \_voffset =\_dimexpr \_pgheight -\_vsize - #6\_tmp \_relax % only bottom margin
              \_fi
      \_else  \_if$#6$\_voffset = #5\_tmp \_relax  % only top margin
              \_else  \_vsize=\_dimexpr \_pgheight - #5\_tmp - #6\_tmp \_relax % top+bottom margin
                      \_voffset = #5\_tmp \_relax
      \_fi\_fi
      \_if 1#1\_shiftoffset=0pt \_def\_prepoffsets{}\_else \_if 2#1% double-page layout
         \_shiftoffset = \_dimexpr \_pgwidth -\_hsize -2\_hoffset \_relax
         \_def\_prepoffsets{\_ifodd\_pageno \_else \_advance\_hoffset \_shiftoffset \_fi
                            \_setpagerightoffset}%
      \_else \_opwarning{use \_string\_margins/1 or \_string\_margins/2}%
   \_fi\_fi\_fi
   \_setpagerightoffset
}
\_def\_setpagedimens{\_isnextchar({\_setpagedimensB}{\_setpagedimensA}}
\_def\_setpagedimensA#1 {\_ifcsname _pgs:#1\_endcsname
   \_ea\_ea\_ea\_setpagedimensB \_csname _pgs:#1\_ea\_endcsname\_space
   \_else \_opwarning{page specification "#1" is undefined}\_fi}
\_def\_setpagedimensB (#1,#2)#3 {\_setpagedimensC\_pgwidth=#1:#3
                                 \_setpagedimensC\_pgheight=#2:#3
        \_pdfpagewidth=\_pgwidth \_pdfpageheight=\_pgheight
}
\_def\_setpagedimensC #1=#2:#3 {#1=#2\_ifx^#3^\_tmp\_else#3\_fi\_relax\_truedimen#1}

\_public \margins ;
```

The common page dimensions are defined here.

margins.opm

```
\_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` [⟨*factor*⟩] does `\mag=`⟨*factor*⟩ and recalculates page dimensions to their true values. `\_truedimen`⟨*dimen-register*⟩ returns true value of ⟨*dimen-register*⟩ regardless of `\mag`.

margins.opm

```
\_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 ;
```

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

margins.opm

```
\_def\_setpagerightoffset{%
   \_pagerightoffset=\_dimexpr\_pdfpagewidth-\_xhsize-\_hoffset-1in\_relax
}
\_setpagerightoffset % setting default value from default values
```

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

```
\_def\_numprint#1{{\_textdir TLT #1}}
```

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

margins.opm

```
\_let\_numprint=\_useit
```

2.20 Colors

2.20.1 Basic concept

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

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

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

Macros `\setcmykcolor{`⟨*C*⟩ ⟨*M*⟩ ⟨*Y*⟩ ⟨*K*⟩`}` or `\setrgbcolor{`⟨*R*⟩ ⟨*G*⟩ ⟨*B*⟩`}` or `\setgreycolor{`⟨*Grey*⟩`}` are used in color selectors. These macros expand to internal `\_setcolor` macro which sets the `\_colorattr` attribute to an integer value and prepares mapping between this value and the real color data. This mapping is used just before each `\shipout` in output routine. The `\_preshipout` pseudo-primitive is used here, it converts attribute values to internal PDF commands for selecting colors.

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

2.20.2 Color mixing

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

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

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

The `\useK` command exactly does:

$$\begin{gathered} k' = \min(C, M, Y), \\ C = (C - k')/(1 - k'),\ \ M = (M - k')/(1 - k'),\ \ Y = (Y - k')/(1 - k'), \\ K = \min(1, K + k'). \end{gathered}$$

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

```
\colordef \Color {\Brown-\Black}
```

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

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

```
\colordef\mycolor{\Grey+.6^\Blue} % the same as \colordef\mycolor{\Grey+.6\Yellow}
```

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

If a CMYK to RGB or RGB to CMYK conversion is needed then direct conversion of given color is used (if declared using `\rgbcmykmap{`⟨*rgb*⟩`}{`⟨*cmyk*⟩`}`) or the following simple formulae are used (ICC profiles are not supported):

$$\begin{gathered} \text{CMYK to RGB:} \\ R = (1 - C)(1 - K),\ \ G = (1 - M)(1 - K),\ \ B = (1 - Y)(1 - K). \\ \text{RGB to CMYK:} \\ K' = \max(R, G, B),\ \ C = (K' - R)/K',\ \ M = (K' - G)/K',\ \ Y = (K' - B)/K',\ \ K = 1 - K'. \end{gathered}$$

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

2.20.3 Implementation

colors.opm

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

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

colors.opm

```
\_def\Blue      {\_setcmykcolor{1 1 0 0}}
\_def\Red       {\_setcmykcolor{0 1 1 0}}
\_def\Brown     {\_setcmykcolor{0 .67 .67 .5}}
\_def\Green     {\_setcmykcolor{1 0 1 0}}
\_def\Yellow    {\_setcmykcolor{0 0 1 0}}
\_def\Cyan      {\_setcmykcolor{1 0 0 0}}
\_def\Magenta   {\_setcmykcolor{0 1 0 0}}
\_def\Grey      {\_setcmykcolor{0 0 0 .5}}
\_def\LightGrey {\_setcmykcolor{0 0 0 .2}}
\_def\White     {\_setgreycolor{1}}
\_def\Black     {\_setgreycolor{0}}
```

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

colors.opm

```
\_def\_setcmykcolor#1{\_setcolor{#1}kK}
\_def\_setrgbcolor#1{\_setcolor{#1}{rg}{RG}}
\_def\_setgreycolor#1{\_setcolor{#1}gG}
\_public \setcmykcolor \setrgbcolor \setgreycolor ;
```

The \onlyrgb declaration redefines \setcmykcolor to do conversion to RGB just before _setcolor is used. The \onlycmyk declaration redefines \setrgbcolor to do conversion to CMYK just before _setcolor is used. Moreover, \onlyrgb re-defines three basic RGB colors for RGB color space and re-declares \colordef as \rgbcolordef.

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 ;
```

The _colorattr for coloring is allocated and _setcolor{⟨*color-data*⟩}{⟨*fill-op*⟩}{⟨*stroke-op*⟩} is defined here. This macro does `\_colorattr=`_colorcnt if the ⟨*color data*⟩ was not used before and prepare mapping from this integer value to the ⟨*color data*⟩ and increments _colorcnt. If the ⟨*color data*⟩ were used already, then _setcolor does `\_colorattr=`⟨*stored-value*⟩. This work is done by the _translatecolor macro. The following mapping macros are created:

`\_color::`⟨*data*⟩ ⟨*fill-op*⟩ `... expands to used` ⟨*attribute-value*⟩
`\_color:`⟨*attribute-value*⟩ `... expands to` ⟨*data*⟩ ⟨*fill-op*⟩
`\_color-s:`⟨*attribute-value*⟩ `... expands to` ⟨*data*⟩ ⟨*stroke-op*⟩

colors.opm

```
\_newattribute \_colorattr
\_newcount \_colorcnt \_colorcnt=1 % allocations start at 1
\_protected\_def\_setcolor{\_colorprefix\_colorattr=\_translatecolor}
\_def\_translatecolor#1#2#3{\_ifcsname _color::#1 #2\_endcsname\_lastnamedcs\_relax
   \_else
      \_colorcnt
      \_sxdef{_color::#1 #2}{\_the\_colorcnt}%
      \_sxdef{_color:\_the\_colorcnt}{#1 #2}%
      \_sxdef{_color-s:\_the\_colorcnt}{#1 #3}%
      \_incr \_colorcnt
   \_fi
}
% Black is the default color.
\_sdef{_color::0 g}{0}
\_sdef{_color:0}{0 g}
\_sdef{_color-s:0}{0 G}
```

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.

colors.opm

```
\_protected\_def \_localcolor   {\_let\_colorprefix=\_relax}
\_protected\_def \_nolocalcolor {\_let\_colorprefix=\_global}
\_public \localcolor \nolocalcolor ;
\_localcolor
```

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`.

colors.opm

```
\_newattribute\_transpattr
\_def\_transparency {\_afterassignment\_transparencyA \_transpattr}
\_def\_transparencyA{%
   \_ifnum\_transpattr<1 \_transpattr=\_noattr \_fi
   \_ifnum\_transpattr>255 \_opwarning{\_noexpand\transparency > 255 not allowed}%
       \_transpattr=\_noattr
   \_else
      \_ifcsname _transp:\_the\_transpattr\_endcsname \_else
         \_edef\_transpv{\_expr{(255-\_the\_transpattr)/255}}%
         \_addextgstate{tr\_the\_transpattr}{<</ca \_transpv\_space /CA \_transpv>>}%
         \_sxdef{_transp:\_the\_transpattr}{}%
         \_ifcsname _transp:0\_endcsname \_else
            \_addextgstate{tr0}{<</ca 1 /CA 1>>}%
            \_sxdef{_transp:0}{}%
         \_fi
      \_fi
   \_fi
}
\_def\_thetransparency{\_ifnum \_transpattr=-"7FFFFFFF 0\_else \_the\_transpattr \_fi}
\_def\_resetattrs{\_colorattr=\_noattr \_transpattr=\_noattr}

\_public \transparency \thetransparency ;
```

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.

colors.opm

```
\_def\_rgbtocmyk #1 #2 #3 ;{\_trycs{_tocmyk:#1 #2 #3}{%
   \_ea \_stripzeros \_detokenize \_ea{\_directlua{
      local kr = math.max(#1,#2,#3)
      if (kr==0) then
         tex.print('0. 0. 0. 1 ;')
      else
         tex.print(string.format('\_pcent.3f \_pcent.3f \_pcent.3f \_pcent.3f ;',
            (kr-#1)/kr, (kr-#2)/kr, (kr-#3)/kr, 1-kr))
      end
}}}}
\_def\_cmyktorgb #1 #2 #3 #4 ;{\_trycs{_torgb:#1 #2 #3 #4}{%
   \_ea \_stripzeros \_detokenize \_ea{\_directlua{
      local kr = 1-#4
      tex.print(string.format('\_pcent.3f \_pcent.3f \_pcent.3f ;',
         (1-#1)*kr, (1-#2)*kr, (1-#3)*kr))
}}}}
\_def\_colorcrop{\_directlua{
   local m=math.max(color_C, color_M, color_Y, color_K)
   if (m>1) then
      color_C=color_C/m  color_M=color_M/m  color_Y=color_Y/m color_K=color_K/m
   end
}}
\_def\_colordefFin{\_colorcrop \_ea \_stripzeros \_detokenize \_ea{\_directlua{
   tex.print(string.format('\_pcent.3f \_pcent.3f \_pcent.3f \_pcent.3f ;',
      color_C, color_M, color_Y, color_K))
}}}
\_def\_douseK{\_colorcrop \_directlua{
   kr=math.min(color_C, color_M, color_Y)
   if (kr>=1) then
      color_C=0  color_M=0  color_Y=0  color_K=1
   else
      color_C=(color_C-kr)/(1-kr)  color_M=(color_M-kr)/(1-kr)
      color_Y=(color_Y-kr)/(1-kr)  color_K=math.min(color_K+kr,1)
   end
}}
```

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`.

colors.opm

```
\_def\_stripzeros #1.#2 #3{\_ifx0#1\_else#1\_fi.\_stripzeroA #2 0 :%
    \_ifx;#3\_else \_space \_ea\_stripzeros\_ea#3\_fi}
\_def\_stripzeroA #10 #2:{\_ifx^#2^\_stripzeroC#1:\_else \_stripzeroB#1 0 :\_fi}
\_def\_stripzeroB #10 #2:{\_ifx^#2^\_stripzeroC#1:\_else #1\_fi}
\_def\_stripzeroC #1 #2:{#1}
```

`\rgbcmykmap` `{`⟨*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.

colors.opm

```
\_def\_rgbcmykmap#1#2{\_sxdef{_torgb:#2}{#1}\_sxdef{_tocmyk:#1}{#2}}
\_public \rgbcmykmap ;
```

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

```
input ⟨data⟩:   ".3\Blue + .6^\KhakiC \useK -\Black"
expanded to:   ".3 !=K 1 1 0 0 +.6^!=R .804 .776 .45 \_useK -!=G 0"
adjusted to:   "\_addcolor .3!=K 1 1 0 0 \_addcolor .6!^R .804 .776 .45
                \_useK \_addcolor -1!=G 0"
and this is processed.
```

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

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

colors.opm

```
\_def\_rgbcolordef  {\_commoncolordef \_setrgbcolor  krg}
\_def\_cmykcolordef {\_commoncolordef \_setcmykcolor KRG}
\_def\_commoncolordef#1#2#3#4#5#6{%
  \_begingroup
     \_directlua{color_C=0 color_M=0 color_Y=0 color_K=0}%
     \_def\_setcmykcolor##1{!=#2 ##1 }%
     \_def\_setrgbcolor ##1{!=#3 ##1 }%
     \_def\_setgreycolor##1{!=#4 ##1 }%
     \_let\_useK=\_relax
     \_edef\_tmpb{+#6}%
     \_replstring\_tmpb{+ }{+}\_replstring\_tmpb{- }{-}%
     \_replstring\_tmpb{+}{\_addcolor}\_replstring\_tmpb{-}{\_addcolor-}%
     \_replstring\_tmpb{^!=}{!^}\_replstring\_tmpb{-!}{-1!}%
     \_ifx K#2\_let\_useK=\_douseK \_fi
     \_tmpb
     \_edef\_tmpb{\_colordefFin}%
     \_ifx k#2\_edef\_tmpb{\_ea\_stripK \_tmpb;}\_fi
  \_ea\_endgroup
  \_ea\_def\_ea#5\_ea{\_ea#1\_ea{\_tmpb}}%
}
\_def\_addcolor#1!#2#3{\_cs{addcolor:#2#3}#1}
\_def\_addcolorA #1 #2 #3 #4 #5 {%
   \_def\_tmpa{#1}\_ifx\_tmpa\_empty \_else \_edef\_tmpa{\_tmpa*}\_fi
   \_directlua{color_C=math.max(color_C+\_tmpa#2,0)
               color_M=math.max(color_M+\_tmpa#3,0)
               color_Y=math.max(color_Y+\_tmpa#4,0)
               color_K=math.max(color_K+\_tmpa#5,0)
}}
\_sdef{addcolor:=K}#1 #2 #3 #4 #5 {\_addcolorA #1 #2 #3 #4 #5 }
\_sdef{addcolor:^K}#1 #2 #3 #4 #5 {\_addcolorA #1 (1-#2) (1-#3) (1-#4) #5 }
\_sdef{addcolor:^G}#1 #2 {\_addcolorA #1 0 0 0 #2 }
\_sdef{addcolor:=G}#1 #2 {\_addcolorA #1 0 0 0 (1-#2) }
\_sdef{addcolor:=R}#1 #2 #3 #4 {%
   \_edef\_tmpa{\_noexpand\_addcolorA #1 \_rgbtocmyk #2 #3 #4 ; }\_tmpa
}
\_sdef{addcolor:^R}#1 #2 #3 #4 {\_cs{addcolor:=R}#1 (1-#2) (1-#3) (1-#4) }

\_sdef{addcolor:=k}#1 #2 #3 #4 #5 {%
   \_edef\_tmpa{\_noexpand\_addcolorA #1 \_cmyktorgb #2 #3 #4 #5 ; 0 }\_tmpa
}
\_sdef{addcolor:^k}#1 #2 #3 #4 #5 {\_cs{addcolor:=k}#1 (1-#2) (1-#3) (1-#4) #5 }
\_sdef{addcolor:^g}#1 #2 {\_addcolorA #1 (1-#2) (1-#2) (1-#2) 0 }
\_sdef{addcolor:=g}#1 #2 {\_addcolorA #1 #2 #2 #2 0 }
\_sdef{addcolor:=r}#1 #2 #3 #4 {\_addcolorA #1 #2 #3 #4 0 }
\_sdef{addcolor:^r}#1 #2 #3 #4 {\_addcolorA #1 (1-#2) (1-#3) (1-#4) 0 }
\_def\_stripK#1 .0;{#1}
\_let\_colordef=\_cmykcolordef  % default \_colordef is \_cmykcolordef
```

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

colors.opm

```
\_def \useK{\_useK}
\_def \colordef {\_colordef}
\_public \cmykcolordef \rgbcolordef ;
```

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

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

colors.opm

```
\_def\_morecolors{%
   \_long\_def\_tmp##1\preparecolorset##2##3##4##5{\_tmpa ##5;,,,;}
   \_def\_tmpa##1,##2,##3,##4;{\_ifx,##1,\_else
```

```
      \_def\_tmpb{##1}\_replstring\_tmpb{1}{}\_replstring\_tmpb{2}{B}%
      \_replstring\_tmpb{3}{C}\_replstring\_tmpb{4}{D}\_replstring\_tmpb{0}{O}%
      \_ifcsname \_tmpb\_endcsname \_else
          \_sdef{\_tmpb}{\_setrgbcolor{##2 ##3 ##4}}\_showcolor\_fi
      \_ea\_tmpa\_fi
   }
   \_ea\_tmp\_input x11nam.def
}
\_let\_showcolor=\_relax % re-define it if you want to print a color catalog
\_public \morecolors ;
```

2.21 The `.ref` file

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

```
\Xrefversion{⟨ref-version⟩}
\_Xpage{⟨gpageno⟩}{⟨pageno⟩}
\_Xtoc{⟨level⟩}{⟨type⟩}{⟨text⟩}{}⟨title⟩
\_Xlabel{⟨label⟩}{⟨text⟩}
\_Xlabel{⟨label⟩}{⟨text⟩}
...
\_Xpage{⟨gpageno⟩}{⟨pageno⟩}
\_Xlabel{⟨label⟩}{⟨text⟩}
...
```

- `\_Xpage` corresponds to the beginning of a page. ⟨*gpageno*⟩ is an internal page number, globally numbered from one. ⟨*pageno*⟩ is the page number (`\the\pageno`) used in pagination (they may differ).
- `\_Xtoc` corresponds to a chapter, section or subsection title on a page. ⟨*title*⟩ is the title of the chapter (⟨*level*⟩=1, ⟨*type*⟩=`chap`), section (⟨*level*⟩=2, ⟨*type*⟩=`sec`) or subsection (⟨*level*⟩=3, ⟨*type*⟩=`secc`).
- `\_Xlabel` corresponds to a labelled object on a page. ⟨*label*⟩ is the label provided by the user in `\label[`⟨*label*⟩`]`, while ⟨*text*⟩ is the text which should be used for the reference (section or table number, for example `2.3.14`).

ref-file.opm

```
\_codedecl \openref {File for references <2021-07-19>} % preloaded in format
```

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.

ref-file.opm

```
\_newwrite\_reffile

\_def\_inputref {%
  \_isfile{\_jobname.ref}\_iftrue
     \_input {\_jobname.ref}%
     \_edef\_prevrefhash{\_mdfive{\_jobname.ref}}%
     \_gfnotenum=0 \_lfnotenum=0 \_mnotenum=0
     \_openref
  \_fi
}
```

`\_mdfive{`⟨*file*⟩`}` expands to the MD5 hash of a given file. We use it to do consistency checking of the `.ref` file. First, we read the MD5 hash of `.ref` file from previous TeX run before it is removed and opened for writing again in the `\_inputref` macro. The hash is saved to `\_prevrefhash`. Second, we read the MD5 hash in the `\_byehook` macro again and if these hashes differ, warning that "ref file has changed" is printed. Try running `optex op-demo` twice to see the effect.

ref-file.opm

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

If the `.ref` file does not exist, then it is not created by default. This means that if you process a document without any forward references then no `\jobname.ref` file is created (it would be unusable). The `\_wref` macro is a dummy in that case.

ref-file.opm

```
\_def\_wrefrelax#1#2{}
\_let\_wref=\_wrefrelax
```

If a macro needs to create and use the `.ref` file, then such macro must first use `\openref`. It creates the file and redefines `\_wref` `\`⟨*macro*⟩`{`⟨*data*⟩`}` so that it saves the line `\`⟨*macro*⟩⟨*data*⟩ to the `.ref` file using the asynchronous `\write` primitive. Finally, `\_openref` destroys itself, because we don't need to open the file again.
`\_wref`⟨*csname*⟩`{`⟨*params*⟩`}` in fact does `\write\_reffile{\string`⟨*csname*⟩⟨*params*⟩`}` and similarly `\_ewref`⟨*csname*⟩`{`⟨*params*⟩`}` does `\write\_reffile{\string`⟨*csname*⟩⟨*expanded-params*⟩`}`.

ref-file.opm

```
\_def\_openref {%
   \_immediate\_openout\_reffile="\_jobname.ref"\_relax
   \_gdef\_wref ##1##2{\_write\_reffile{\_bslash\_csstring##1##2}}%
   \_immediate\_write\_reffile {\_pcent\_pcent\_space OpTeX <\_optexversion> - REF file}%
   \_immediate\_wref \Xrefversion{{\_REFversion}}%
   \_ifx\_refdecldata\_empty \_else \_refdeclwrite \_fi
   \_gdef\_openref{}%
}
\_def\_ewref #1#2{\_edef\_ewrefA{#2}\_ea\_wref\_ea#1\_ea{\_ewrefA}}
\_def\openref{\_openref}
```

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

`\Xrefversion{`⟨*version*⟩`}`

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

ref-file.opm

```
\_def\_REFversion{6} % current version of .ref files in OpTeX
\_def\_Xrefversion#1{\_ifnum #1=\_REFversion\_relax \_else \_endinput \_fi}
\_public \Xrefversion ; % we want to ignore .ref files generated by OPmac
```

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

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

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

ref-file.opm

```
\_def\_refdecldata{}
\_def\_refdecl{\_bgroup \_catcode`=12 \_catcode`=12 \_catcode`=12 \_refdeclA}
\_def\_refdeclA#1{\_egroup
   \_ifx\_refdecldata\_empty\_else \_global\_addto\_refdecldata{^^J}\_fi
   \_global\_addto\_refdecldata{#1}%
   \_ifx\_openref\_empty \_refdeclwrite \_fi
}
\_def\_refdeclwrite{%
   \_immediate\_write\_reffile{\_pcent\_space \_string\refdecl:^^J\_detokenize\_ea{\_refdecldata}}%
   \_gdef\_refdecldata{}%
}
\_public \refdecl ;
```

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.

references.opm

```
\_codedecl \ref {References <2023-07-03>} % preloaded in format
```

`\_Xpage` `{`⟨*gpageno*⟩`}{`⟨*pageno*⟩`}` saves the parameter pair into `\_currpage`. Resets `\_lfnotenum`; it is used if footnotes are numbered from one at each page.

references.opm

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

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}
```

references.opm

```
\_newcount\_unresolvedrefs
\_unresolvedrefs=0
```

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

references.opm

```
\_def\_Xlabel#1#2{\_sdef{_lab:#1}{#2}\_sxdef{_pgref:#1}{\_currpage}}
```

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

references.opm

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

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

references.opm

```
\_def\_ref[#1]{\_xdef\_lastreflabel{#1}\_isnextchar\_bgroup{\_refA}{\_refA{@}}}
\_def\_refA #1{\_isdefined{_lab:\_lastreflabel}%
  \_iftrue \_ilink[ref:\_lastreflabel]{\_reftext{\_csname _lab:\_lastreflabel\_endcsname}{#1}}%
  \_else \_reftext{??}{#1}\_opwarning{label [\_lastreflabel] unknown. Try to TeX me again}%
     \_incr\_unresolvedrefs \_openref
  \_fi
}
\_def\_pgref[#1]{\_xdef\_lastreflabel{#1}\_isnextchar\_bgroup{\_pgrefA}{\_pgrefA{@}}}
\_def\_pgrefA #1{\_isdefined{_pgref:\_lastreflabel}%
```

```
  \_iftrue \_ea\_ea\_ea\_pgrefB \_csname _pgref:\_lastreflabel\_endcsname{#1}%
  \_else \_reftext{??}{#1}\_opwarning{pg-label [\_lastreflabel] unknown. Try to TeX me again}%
  \_incr\_unresolvedrefs \_openref
  \_fi
}
\_def\_pgrefB #1#2#3{\_ilink[pg:#1]{\_reftext{#2}{#3}}}

\_public \ref \pgref ;
```

`\_reftext`{⟨*implicit-text*⟩}{⟨*given-text*⟩} expands to the ⟨*given-text*⟩ but the optional `@` in the ⟨*given-text*⟩ is replaced by the ⟨*implicit-text*⟩ first.

`references.opm`

```
\_def\_reftext  #1#2{\_isatin #2@\_iffalse \_numprint{#2}\_else\_reftextA{#1}#2\_fin \_fi}
\_def\_reftextA #1#2@#3\_fin {#2\_numprint{#1}#3}
\_def\_isatin #1@#2\_iffalse {\_ifx\_fin#2\_fin}
```

Default `\_printlabel` is empty macro (labels are not printed). The `\showlabels` redefines it as box with zero dimensions and with left lapped `[`⟨*label*⟩`]` in blue 10pt `\tt` font shifted up by 1.7ex. The color of labels is set by `\_labelcolor` (default is RGB blue).

`references.opm`

```
\_def\_printlabel#1{}
\_def\_labelcolor{\_setrgbcolor{0 0 1}}
\_def\_showlabels {%
   \_def\_printlabel##1{\_vbox to\_zo{\_vss\_llap{\_labelcolor\_labelfont[##1]}\_kern1.7ex}}%
   \_fontdef\_labelfont{\_setfontsize{at10pt}\_tt}
}
\_public \showlabels ;
```

2.23 Hyperlinks

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

- `ref:`⟨*label*⟩ – the destination is created when `\label[`⟨*label*⟩`]` is used, see also the section 2.22.
- `toc:`⟨*tocrefnum*⟩ – the destination is created at chap/sec/secc titles, see also the section 2.24.
- `pg:`⟨*gpageno*⟩ – the destination is created at beginning of each page, see also the section 2.18.
- `cite:`⟨*bibpart*⟩`/`⟨*bibnum*⟩ – the destination is created in bibliography reference, see section 2.32.1.
- `fnt:`⟨*gfnotenum*⟩ – link form text to footnote, see also section 2.34.
- `fnf:`⟨*gfnotenum*⟩ – link from footnote to text, see also section 2.34.
- `url:`⟨*url*⟩ – used by `\url` or `\ulink`, see also the end of this section.

The ⟨*tocrefnum*⟩, ⟨*gpageno*⟩, ⟨*bibnum*⟩, and ⟨*gfnotenum*⟩ are numbers starting from one and globally incremented by one in the whole document. The registers `\tocrefnum`, `\gpageno`, `\bibnum`, and `\_gfnotenum` are used for these numbers.

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

The color for active links can be declared by `\def\_`⟨*type*⟩`linkcolor`, the border around link can be declared by `\def\_`⟨*type*⟩`border`. These macros are not declared by default, so color for active links are given only by `\hyperlinks` macro and borders are invisible. For example `\def\_toclinkcolor{\Red}` means that links from table of contents are in red. Another example `\def\_tocborder{1 0 0}` causes red frames in TOC (not printed, only visible in PDF viewers).

`hyperlinks.opm`

```
\_codedecl \ulink {Hyperlinks <2021-08-31>} % preloaded in format
```

`\dest[`⟨*type*⟩`:`⟨*spec*⟩`]` creates a destination of internal links. The destination is declared in the format ⟨*type*⟩`:`⟨*spec*⟩. If the `\hyperlinks` command in not used, then `\dest` does nothing else it is set to `\_destactive`. The `\_destactive` is implemented by `\_pdfdest` primitive. It creates a box using `\_destbox[`⟨*type*⟩`:`⟨*spec*⟩`]` in which the destination is shifted by `\_destheight`. The reason is that the destination is exactly at the top border of the PDF viewer but we want to see the line where the destination is. The destination box is positioned differently dependent on the current vertical or horizontal mode.

hyperlinks.opm

```
\_def\_destheight{1.4em}
\_def\_destactive[#1:#2]{\_if$#2$\_else\_ifvmode
      \_tmpdim=\_prevdepth \_prevdepth=-1000pt
      \_destbox[#1:#2]\_prevdepth=\_tmpdim
   \_else \_destbox[#1:#2]%
   \_fi\_fi
}
\_def\_destbox[#1]{\_vbox to\_zo{\_kern-\_destheight \_pdfdest name{#1} xyz\_vss}}
\_def\_dest[#1]{}
\_public \dest ;
```

Each hyperlink is created internally by `\_xlink`{⟨*type*⟩}{⟨*spec*⟩}{⟨*color*⟩}{⟨*text*⟩}. This macro expands to `\_quitvmode`{⟨*text*⟩} by default, i.e. no active hyperlink is created, only ⟨*text*⟩ is printed in horizontal mode (and in a group). If `\hyperlinks` is used, then `\_xlink` gets the meaning of `\_xlinkactive` and hyperlinks are created by the `\pdfstartlink`/`\pdfendlink` primitives. The ⟨*text*⟩ has given ⟨*color*⟩ only when hyperlink is created. If `\_`⟨*type*⟩`linkcolor` is defined, it has precedence over ⟨*color*⟩.
The `\_linkdimens` macro declares the dimensions of link area.
A specific action can be defined for each link ⟨*type*⟩ by the macro `\_`⟨*type*⟩`action`{⟨*spec*⟩}. OpTeX defines only `\_urlaction`{⟨*url*⟩}. The default link action (when `\_`⟨*type*⟩`action` is not defined) is `goto name`{⟨*type*⟩:⟨*spec*⟩} (an internal link). It is declared in the `\_linkactions`{⟨*type*⟩}{⟨*spec*⟩} macro. The `\_pdfstartlink` primitive uses `attr{\_pdfborder`{⟨*type*⟩}}. The `\_pdfborder`{⟨*type*⟩} macro expands to `/C[? ? ?] /Border[0 0 .6]` if the `\_`⟨*type*⟩`border` macro (i.e. `\_refborder`, `\_citeborder`, `\_tocborder`, `\_pgborder`, `\_urlborder`, `\_fntborder` or `\_fnfborder`) is defined.

hyperlinks.opm

```
\_protected\_def\_xlinkactive#1#2#3#4{\_quitvmode
   \_pdfstartlink \_linkdimens attr{\_pdfborder{#1}}\_linkactions{#1}{#2}\_relax
   {\_localcolor\_trycs{_#1linkcolor}{#3}#4}\_pdfendlink
}
\_protected\_def\_xlink#1#2#3#4{\_quitvmode{#4}}

\_def\_linkdimens{height.9em depth.3em}

\_def\_linkactions#1#2{\_ifcsname _#1action\_endcsname
   \_lastnamedcs{#2}\_else goto name{#1:#2}\_fi}
\_def\_urlaction #1{user{/Subtype/Link/A <</Type/Action/S/URI/URI(#1)>>}}

\_def\_pdfborder#1{\_ifcsname _#1border\_endcsname
     /C [\_csname _#1border\_endcsname] /Border [0 0 .6]\_else  /Border [0 0 0]\_fi
}
```

`\link`[⟨*type*⟩:⟨*spec*⟩]{⟨*color*⟩}{⟨*text*⟩} creates a link. It is kept here for backward compatibility and it is equivalent to `\_xlink`{⟨*type*⟩}{⟨*spec*⟩}{⟨*color*⟩}{⟨*text*⟩}. If `\_`⟨*type*⟩`action` is not defined then `\link` creates internal link do the `\dest`[⟨*type*⟩:⟨*spec*⟩]. You can have more links with the same ⟨*type*⟩:⟨*spec*⟩ but only one `\dest` in the document.
`\ilink`[⟨*type*⟩:⟨*spec*⟩]{⟨*text*⟩} is equivalent to `\link` but the ⟨*color*⟩ is used from `\hyperlinks` declaration (or it is overwritten by `\def\_`⟨*type*⟩`linkcolor`).
`\ulink`[⟨*url*⟩]{⟨*text*⟩} creates external link. The ⟨*url*⟩ is detokenized with `\escapechar=-1` before it is used, so `\%`, `\#` etc. can be used in the ⟨*url*⟩.

hyperlinks.opm

```
\_def\_link[#1:#2]{\_xlink{#1}{#2}}
\_def\_ilink[#1:#2]#3{\_xlink{#1}{#2}\_ilinkcolor{#3}}
\_def\_ulink[#1]#2{{\_escapechar=-1 \_ea}\_expanded
   {\_noexpand\_xlink{url}{\_detokenize{#1}}}\_elinkcolor{#2}}

\_public \ilink \ulink \link ;
```

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

hyperlinks.opm

```
\_def\_hyperlinks#1#2{%
   \_let\_dest=\_destactive \_let\_xlink=\_xlinkactive
   \_let\_ilinkcolor=#1\_empty
   \_let\_elinkcolor=#2\_empty
   \_public \dest \xlink ;%
}
\_public \hyperlinks ;
```

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

hyperlinks.opm

```
\_def\_url#1{{%
   \_def\_tmpa{#1}\_replstring\_tmpa {}{}%
   \_def\_tmpb{#1}\_replstring\_tmpb {}{[||]}%
   {\_escapechar=-1 \_ea}\_ea\_edef\_ea\_tmpb\_ea{\_detokenize\_ea{\_tmpb}}%
   \_replstring\_tmpb{[||]}{{gb|}}%
   \_replstring\_tmpb{ }{{ }}%
   \_replstring\_tmpb{://}{{://}}%
   \_ea\_ulink \_ea[\_ea{\_tmpa}] {\_urlfont \_textdirection=0 \_ea\_urlA\_tmpb\_fin}%
}}
\_def\_urlA#1{\_ifx\_fin#1\_else \_urlC{}{#1}\_fi}
\_def\_urlB#1{\_ifx\_fin#1\_else \_urlC{\_urlxskip}{#1}\_fi}
\_def\_urlC#1#2{%
   \_ifcsname _ur:#2\_endcsname \_lastnamedcs \_ea\_ea\_ea \_urlA
   \_else #1#2\_ea\_ea\_ea \_urlB \_fi
}
\_sdef{_ur:://}{\_urlskip:\_urlskip/\_urlskip/\_urlbskip}
\_sdef{_ur:/}{\_urlskip/\_urlbskip}
\_sdef{_ur:.}{\_urlskip.\_urlbskip}
\_sdef{_ur:?}{\_urlskip?\_urlbskip}
\_sdef{_ur:=}{\_urlskip=\_urlbskip}
\_sdef{_ur:-}{\_urlskip-\_urlbskip}
\_sdef{_ur:&}{\_urlskip\_char`\_urlbskip}
\_sdef{_ur:gb|}{\_urlgskip}

\_def\_urlfont{\_tt}                    % url font
\_def\_urlxskip{\_penalty9990\_hskip0pt plus0.03em\_relax} % skip between normal characters
\_def\_urlskip{\_null\_nobreak\_hskip0pt plus0.1em\_relax} % skip before :// / . ? = - &
\_def\_urlbskip{\_penalty100 \_hskip0pt plus0.1em\_relax}  % skip after  :// / . ? = - &
\_def\_urlgskip{\_penalty-500\_relax}   % "goodbreak" penalty generated by

\_public \url ;
```

2.24 Making table of contents

maketoc.opm

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

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

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

The last two parameters are restored from previous _Xpage{⟨*pageno*⟩}{⟨*gpageno*⟩}, data were saved in the _currpage macro.

We read the ⟨*title*⟩ parameter by \scantoeol from .ref file because the ⟨*title*⟩ can include something like `{`.

maketoc.opm

```
\_def\_toclist{}
\_newifi \_ifischap \_ischapfalse

\_def\_Xtoc#1#2#3#4{\_ifnum#1=0 \_ischaptrue\_fi
   \_addto\_toclist{\_tocline{#1}{#2}{#3}{#4}}\_scantoeol\_XtocA}
\_def\_XtocA#1{\_addto\_toclist{{#1}}\_ea\_addto\_ea\_toclist\_ea{\_currpage}}
```

_tocline{⟨*level*⟩}{⟨*type*⟩}{⟨*number*⟩}{⟨*o-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_iindent_null _par. This causes that the last line of the record is shifted outside the margin given by _rightskip. A typical record (with long ⟨*title*⟩) looks like this:

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

maketoc.opm

```
\_newcount \_tocrefnum
\_def\_tocline#1#2#3#4#5#6#7{%
   \_advance\_tocrefnum by1
   \_bgroup
      \_leftskip=\_iindent \_rightskip=2\_iindent
      \_ifischap \_advance\_leftskip by \_iindent \_fi
      \_def\_pgn##1{\_ilink[pg:#6]{\_numprint{##1}}}%
      \_the\_everytocline
      \_ifcsname _tocl:#1\_endcsname
         \_cs{_tocl:#1}{#3}{\_scantextokens{#5}}{#7}\_par
      \_fi
   \_egroup
}
\_public \tocrefnum ;
```

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

maketoc.opm

```
\_sdef{_tocl:1}#1#2#3{\_nofirst\_bigskip
   \_bf\_llaptoclink{#1}{#2}\_nobreak\_hfill \_pgn{#3}\_tocpar}
\_sdef{_tocl:2}#1#2#3{\_llaptoclink{#1}{#2}\_tocdotfill \_pgn{#3}\_tocpar}
\_sdef{_tocl:3}#1#2#3{\_advance\_leftskip by\_iindent \_cs{_tocl:2}{#1}{#2}{#3}}
```

The auxiliary macros are:

- _llaptoclink⟨*text*⟩ does _noindent_llap{⟨*linked text*⟩}.
- _tocdotfill creates dots in the TOC.
- _nofirst\macro applies the \macro only if we don't print the first record of the TOC.
- _tocpar finalizes one TOC record with rlapped ⟨*pageno*⟩.
- _pgn{⟨*pageno*⟩} creates ⟨*pageno*⟩ as link to real ⟨*gpage*⟩ saved in #6 of _tocline. This is temporarily defined in the _tocline.

maketoc.opm

```
\_def\_llaptoclink#1{\_noindent
   \_llap{\_ilink[toc:\_the\_tocrefnum]{\_enspace\_numprint{#1}\_kern.4em}\_kern.1em}}
\_def\_tocdotfill{\_nobreak\_leaders\_hbox to.8em{\_hss.\_hss}\_hskip 1em plus1fill\_relax}
\_def\_nofirst #1{\_ifnum \_lastpenalty=11333 \_else #1\_fi}
\_def\_tocpar{\_nobreak \_hskip-2\_iindent\_null \_par}
```

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

```
\def\addtotoc#1#2#3#4#5{%
   \incr\_tocrefnum
   \_dest[toc:\_the\_tocrefnum]%
   \_ewref\_Xtoc{{#1}{#2}{#3}{#4}#5}%
}
```

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

```
\sdef{_tocl:10}#1#2#3{\medskip\hbox{\Blue #2}\medskip}
```

Now, users can add a blue line into TOC by

```
\addtotoc{10}{blue-line}{}{\relax}{⟨blue text to be added in the TOC⟩}
```

anywhere in the document. Note that `\relax` in the fourth parameter means that outline will be not generated. And second parameter `blue-line` is only a comment (unused in macros).

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

maketoc.opm

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

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

maketoc.opm

```
\_newtoks \_regtoc  \_newtoks \_regmark  \_newtoks \_regoul

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

2.25 PDF outlines

2.25.1 Nesting PDF outlines

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

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

Each outline is mapped to the control sequence of the type `\_ol:`⟨*num*⟩ or `\_ol:`⟨*num*⟩`:`⟨*num*⟩ or `\_ol:`⟨*num*⟩`:`⟨*num*⟩`:`⟨*num*⟩ or etc. The first one is reserved for level 0, the second one for level 1 (chapters), the third one for level 2 (sections) etc. The number of direct descendants will be stored in these macros after the first step is finished. Each new outline of a given level increases the ⟨*num*⟩ at the given level. When the first step is processed then (above that) the `\_ol:..` sequence of the parent increases its value too. The `_ol:...` sequences are implemented by `\_ol:\_count0:\_count1:\count2` etc. For example, when section (level 2) is processed in the first step then we do:

```
\advance \count2 by 1
                 % increases the mapping pointer of the type
                 % \_ol:\_count0:\_count1:\_count2 of this section
\advance \_ol:\_count0:\_count1 by 1
                 % increases the number of descendants connected
                 % to the parent of this section.
```

When the second step is processed, then we only read the stored data about the number of descendants. And we use it in `count` parameter of `\_pdfoutline` primitive.

For linking, we use the same links as in TOC, i.e. the `toc:\_the\_tocrefnum` labels are used.

`\insertoutline` `{`⟨*text*⟩`}` inserts one outline with zero direct descendants. It creates a link destination of the type `oul:`⟨*num*⟩ into the document (where `\insertoutline` is used) and the link itself is created too in the outline.

`outlines.opm`

```
\_codedecl \outlines {PDF outlines <2021-02-09>} % preloaded in format

\_def\_outlines#1{\_pdfcatalog{/PageMode/UseOutlines}\_openref
   \_ifx\_toclist\_empty
     \_opwarning{\_noexpand\outlines -- data unavailable. TeX me again}%
     \_incr\_unresolvedrefs
   \_else
     \_ifx\_dest\_destactive \_else
        \_opwarning{\_noexpand\outlines doesn't work when \_noexpand\hyperlinks isn't declared}\_fi
     {\_let\_tocline=\_outlinesA
      \_count0=0 \_count1=0 \_count2=0 \_count3=0 \_toclist % calculate numbers o childs
      \_def\_outlinelevel{#1}\_let\_tocline=\_outlinesB
      \_tocrefnum=0 \_count0=0 \_count1=0 \_count2=0 \_count3=0
      \_toclist}% create outlines
   \_fi
}
\_def\_outlinesA#1#2#3#4#5#6#7{%
   \_isequal{\relax}{#4}\_iffalse
      \_advance\_count#1 by1
      \_ifcase#1\_or
         \_addoneol{_ol:\_the\_count0}\_or
         \_addoneol{_ol:\_the\_count0:\_the\_count1}\_or
         \_addoneol{_ol:\_the\_count0:\_the\_count1:\_the\_count2}\_or
         \_addoneol{_ol:\_the\_count0:\_the\_count1:\_the\_count2:\_the\_count3}\_fi
   \_fi
}
\_def\_addoneol#1{%
   \_ifcsname #1\_endcsname
           \_tmpnum=\_csname#1\_endcsname\_relax
           \_advance\_tmpnum by1 \_sxdef{#1}{\_the\_tmpnum}%
   \_else \_sxdef{#1}{1}%
   \_fi
}
\_def\_outlinesB#1#2#3#4#5#6#7{%
   \_advance\_tocrefnum by1
   \_isequal{\relax}{#4}\_iffalse
      \_advance\_count#1 by1
      \_ifcase#1%
         \_tmpnum=\_trycs{_ol:\_the\_count0}{0}\_or
         \_tmpnum=\_trycs{_ol:\_the\_count0:\_the\_count1}{0}\_relax\_or
         \_tmpnum=\_trycs{_ol:\_the\_count0:\_the\_count1:\_the\_count2}{0}\_relax\_or
         \_tmpnum=\_trycs{_ol:\_the\_count0:\_the\_count1:\_the\_count2:\_the\_count3}{0}\_relax\_or
         \_tmpnum = 0\_relax\_fi
      \_isempty{#4}\_iftrue \_pdfunidef\_tmp{#5}\_else \_pdfunidef\_tmp{#4}\_fi
      \_outlinesC{toc:\_the\_tocrefnum}{\_ifnum#1<\_outlinelevel\_space\_else-\_fi}{\_tmpnum}{\_tmp}%
   \_fi
}
\_def\_outlinesC#1#2#3#4{\_pdfoutline goto name{#1} count #2#3{#4}\_relax}

\_newcount\_oulnum
\_def\_insertoutline#1{\_incr\_oulnum
   \_pdfdest name{oul:\_the\_oulnum} xyz\_relax
   \_pdfunidef\_tmp{#1}%
   \_pdfoutline goto name{oul:\_the\_oulnum} count0 {\_tmp}\_relax
}
\_public \outlines \insertoutline ;
```

2.25.2 Strings in PDF outlines

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

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

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

This example shows a hexadecimal PDF string (enclosed in `<>` as opposed to the literal PDF string enclosed in `()`). In these strings each byte is represented by two hexadecimal characters (`0-9`, `A-F`). You can tell the encoding is UTF-16BE, because it starts with "Byte order mark" `FEFF`. Each unicode character is then encoded in one or two byte pairs. The example string corresponds to the text "Cvičení 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 conversion is done by the expandable "pseudoprimitive" `\pdfstring`.

pdfuni-string.opm

```
\_codedecl \pdfunidef {PDFunicode strings for outlines <2024-12-18>} % preloaded in format
```

`\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, and `\pdfstring` is applied.
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.

pdfuni-string.opm

```
\_def\_pdfunidef#1#2{%
   \_begingroup
      \_catcodetable\_optexcatcodes \_adef"{"}\_adef'{'}%
      \_the\_regoul \_relax % \_regmacro alternatives of logos etc.
      \_ifx\_savedttchar\_undefined \_def#1{\_scantextokens{\_unexpanded{#2}}}%
      \_else \_lccode`=\_savedttchar \_lowercase{\_prepinverb#1;}{#2}\fi
      \_edef#1{#1}%
      \_escapechar=-1
      \_edef#1{#1\_empty}%
      \_escapechar=`
      \_ea\_edef \_ea#1\_ea{\_ea\_removeoutmath   #1$\_fin$}%  $x$ -> x
      \_ea\_edef \_ea#1\_ea{\_ea\_removeoutbraces #1{\_fin}}%  {x} -> x
      \_edef#1{\_detokenize\_ea{#1}}%
      \_ea
   \_endgroup
   \_ea\_edef\_ea#1\_ea{\_pdfstring\_ea{#1}}
}
\_def\_removeoutbraces #1#{#1\_removeoutbracesA}
\_def\_removeoutbracesA #1{\_ifx\_fin#1\_else #1\_ea\_removeoutbraces\_fi}
\_def\_removeoutmath #1$#2${#1\_ifx\_fin#2\_else #2\_ea\_removeoutmath\_fi}
```

The `\_prepinverb`⟨*macro*⟩⟨*separator*⟩`{`⟨*text*⟩`}`, e.g. `\_prepinverb\tmpb|{aaa |bbb| cccc |dd| ee}` does `\def\tmpb{`⟨*su*⟩`{aaa }bbb`⟨*su*⟩`{ cccc }dd`⟨*su*⟩`{ ee}}` where ⟨*su*⟩ is `\scantextokens\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.

pdfuni-string.opm

```
\_def\_prepinverb#1#2#3{\_def#1{}%
   \_def\_dotmpb ##1#2##2{\_addto#1{\_scantextokens{\_unexpanded{##1}}}%
      \_ifx\_fin##2\_else\_ea\_dotmpbA\_ea##2\_fi}%
   \_def\_dotmpbA ##1#2{\_addto#1{##1}\_dotmpb}%
   \_dotmpb#3#2\_fin
}
```

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.

pdfuni-string.opm

```
\_regmacro {}{}{\_let\em=\_empty \_let\rm=\_empty \_let\bf=\_empty
    \_let\it=\_empty \_let\bi=\_empty \_let\tt=\_empty \_let=\_empty
    \_let~=\_space
}
\public \pdfunidef ;
```

2.26 Chapters, sections, subsections

sections.opm

```
\_codedecl \chap {Titles, chapters, sections, subsections <2024-01-19>} % preloaded in format
```

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.

sections.opm

```
\_def \_titfont  {\_scalemain\_typoscale[\_magstep4/\_magstep5]\_boldify}
\_def \_chapfont {\_scalemain\_typoscale[\_magstep3/\_magstep3]\_boldify}
\_def \_secfont  {\_scalemain\_typoscale[\_magstep2/\_magstep2]\_boldify}
\_def \_seccfont {\_scalemain\_typoscale[\_magstep1/\_magstep1]\_boldify}
```

The `\tit` macro 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.

sections.opm

```
\_def\_printtit #1{\_vglue\_titskip
  {\_leftskip=0pt plus1fill \_rightskip=\_leftskip % centering
   \_titfont \_noindent \_scantextokens{#1}\_par}%
   \_nobreak\_bigskip
}
\_def\_tit{\_scantoeol\_printtit}
\_sdef{_eol:tit}{\_printtit} % enables \bracedparam\tit{title}

\_public \tit ;
```

You can re-define `\_printchap`, `\_printsec` or `\_printsecc` macros if another design of section titles is needed. These macros get the ⟨*title*⟩ text in its parameter. The common recommendations for these macros are:

- Use `\_abovetitle{`⟨*penaltyA*⟩`}{`⟨*skipA*⟩`}` and `\_belowtitle{`⟨*skipB*⟩`}` for inserting vertical material above and below the section title. The arguments of these macros are normally used, i. e. `\_abovetitle` inserts ⟨*penaltyA*⟩⟨*skipA*⟩ and `\_belowtitle` inserts ⟨*skipB*⟩. But there is an exception: if `\_belowtitle{`⟨*skipB*⟩`}` is immediately followed by `\_abovetitle{`⟨*penaltyA*⟩`}{`⟨*skipA*⟩`}` (for example section title is immediately followed by subsection title), then only ⟨*skipA*⟩ is generated, i. e. ⟨*skipB*⟩⟨*penaltyA*⟩⟨*skipA*⟩ is reduced only to ⟨*skipA*⟩. The reason for such behavior: we don't want to duplicate vertical skip and we don't want to use the negative penalty in such cases. Moreover, `\_abovetitle{`⟨*penaltyA*⟩`}{`⟨*skipA*⟩`}` takes previous whatever vertical skip (other than from `\_belowtitle`) and generates only greater from this pair of skips. It means that ⟨*whatever-skip*⟩⟨*penaltyA*⟩⟨*skipA*⟩ is transformed to ⟨*penaltyA*⟩max(⟨*whatever-skip*⟩⟨*skipA*⟩). The reason for such behavior: we don't want to duplicate vertical skips (from `\_belowlistskip`, for example) above the title.
- Use `\_printrefnum[`⟨*pre*⟩`@`⟨*post*⟩`]` in horizontal mode. It prints ⟨*pre*⟩⟨*ref-num*⟩⟨*post*⟩. The ⟨*ref-num*⟩ is `\_thechapnum` or `\_thesecnum` or `\_theseccnum` depending on what type o title is processed. If `\nonum` prefix is used then `\_printrefnum` prints nothing. The macro `\_printrefnum` does more work: it creates destination of hyperlinks (if `\hyperlinks{}{}` is used) and saves references from the label (if `\label[`⟨*label*⟩`]` precedes) and saves references for the table of contents (if `\maketoc` is used).
- Use `\nbpar` for closing the paragraph for printing title. This command inserts `\_nobreak` between each line of such paragraph, so the title cannot be broken into more pages.
- You can use `\_firstnoindent` in order to the first paragraph after the title is not indented.

sections.opm

```
\_def\_printchap #1{\_vfill\_supereject \_prevdepth=0pt
   \_vglue\_medskipamount % shifted by topkip+\medskipamount
   {\_chapfont \_noindent \_mtext{chap} \_printrefnum[@]\_par
    \_nobreak\_smallskip
    \_noindent \_raggedright #1\_nbpar}\_mark{}%
   \_nobreak \_belowtitle{\_bigskip}%
   \_firstnoindent
}
\_def\_printsec#1{\_par
   \_abovetitle{\_penalty-151}\_bigskip
   {\_secfont \_noindent \_raggedright \_printrefnum[@\_quad]#1\_nbpar}\_insertmark{#1}%
   \_nobreak \_belowtitle{\_medskip}%
   \_firstnoindent
}
\_def\_printsecc#1{\_par
   \_abovetitle{\_penalty-101}{\_medskip\_smallskip}
   {\_seccfont \_noindent \_raggedright \_printrefnum[@\_quad]#1\_nbpar}%
   \_nobreak \_belowtitle{\_medskip}%
   \_firstnoindent
}
```

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)

sections.opm

```
\_newcount\_sectionlevel
\_def \_secinfo {\_ifcase \_sectionlevel
   part\_or chap\_or sec\_or secc\_or seccc\_fi
}
```

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 `\_othe\_chapnum.` expands to empty. Sections have numbers ⟨*num*⟩ and subsections ⟨*num*⟩.⟨*num*⟩. On the other hand, if chapter is used in the document then `\_chapnum>0` and sections have numbers ⟨*num*⟩`.`⟨*num*⟩ and subsections have numbers ⟨*num*⟩`.`⟨*num*⟩`.`⟨*num*⟩.

sections.opm

```
\_newcount \_chapnum  % chapters
\_newcount \_secnum   % sections
\_newcount \_seccnum  % subsections
\_newcount \_tnum     % table numbers
\_newcount \_fnum     % figure numbers
\_newcount \_dnum     % numbered display maths

\_def \_chapx {\_secx  \_secnum=0  \_lfnotenum=0 }
\_def \_secx  {\_seccx \_seccnum=0 \_tnum=0 \_fnum=0 \_dnum=0 \_resetABCDE }
\_def \_seccx {}

\_def \_thechapnum {\_the\_chapnum}
\_def \_thesecnum  {\_othe\_chapnum.\_the\_secnum}
\_def \_theseccnum {\_othe\_chapnum.\_the\_secnum.\_the\_seccnum}
\_def \_thetnum    {\_othe\_chapnum.\_othe\_secnum.\_the\_tnum}
\_def \_thefnum    {\_othe\_chapnum.\_othe\_secnum.\_the\_fnum}
\_def \_thednum    {(\_the\_dnum)}

\_def\_othe #1.{\_ifnum#1>0 \_the#1.\_fi}
```

The `\notoc` and `\nonum` prefixes are implemented by internal `\_ifnotoc` and `\_ifnonum`. They are reset after each chapter/section/subsection by the `\_resetnonumnotoc` macro.

sections.opm

```
\_newifi \_ifnotoc  \_notocfalse  \_def\_notoc {\_global\_notoctrue}
\_newifi \_ifnonum  \_nonumfalse  \_def\_nonum {\_global\_nonumtrue}
\_def \_resetnonumnotoc{\_global\_notocfalse \_global\_nonumfalse}
\_public \notoc \nonum ;
```

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

sections.opm

```
\_optdef\_chap[]{\_trylabel \_scantoeol\_inchap}
\_optdef\_sec []{\_trylabel \_scantoeol\_insec}
\_optdef\_secc[]{\_trylabel \_scantoeol\_insecc}
\_def\_trylabel{\_istoksempty\_opt\_iffalse \_label[\_the\_opt]\_fi}

\_sdef{_eol:chap}{\_inchap} % enables \bracedparam\chap{title}
\_sdef{_eol:sec}{\_insec}   % enables \bracedparam\sec{title}
\_sdef{_eol:secc}{\_insecc} % enables \bracedparam\secc{title}

\_def\_inchap #1{\_par \_sectionlevel=1
   \_def \_savedtitle {#1}% saved to .ref file
   \_ifnonum \_else {\_globaldefs=1 \_incr\_chapnum \_chapx}\_fi
   \_edef \_therefnum {\_ifnonum \_space \_else \_thechapnum \_fi}%
   \_printchap{\_scantextokens{#1}}%
   \_resetnonumnotoc
}
\_def\_insec #1{\_par \_sectionlevel=2
   \_def \_savedtitle {#1}% saved to .ref file
   \_ifnonum \_else {\_globaldefs=1 \_incr\_secnum \_secx}\_fi
   \_edef \_therefnum {\_ifnonum \_space \_else \_thesecnum \_fi}%
   \_printsec{\_scantextokens{#1}}%
   \_resetnonumnotoc
}
\_def\_insecc #1{\_par \_sectionlevel=3
   \_def \_savedtitle {#1}% saved to .ref file
   \_ifnonum \_else {\_globaldefs=1 \_incr\_seccnum \_seccx}\_fi
   \_edef \_therefnum {\_ifnonum \_space \_else \_theseccnum \_fi}%
   \_printsecc{\_scantextokens{#1}}%
   \_resetnonumnotoc
}
\_public \chap \sec \secc ;
```

The `\_printrefnum`[⟨*pre*⟩`@`⟨*post*⟩] macro is used in `\_print*` macros.

Note that the ⟨*tite-text*⟩ is `\detokenize`d before `\_wref`, so the problem of “fragile macros” from old 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.

sections.opm

```
\_def \_printrefnum [#1@#2]{\_leavevmode % we must be in horizontal mode
   \_ifnonum \_else #1\_numprint\_therefnum #2\_fi
   \_wlabel \_therefnum  % references, if `\label[<label>]` is declared
   \_ifnotoc \_else \_incr \_tocrefnum
       \_dest[toc:\_the\_tocrefnum]%
       \_ewref\_Xtoc{{\_the\_sectionlevel}{\_secinfo}%
                     {\_therefnum}{\_theoutline}\_detokenize\_ea{\_savedtitle}}%
   \_fi
   \_gdef\_theoutline{}%
}
```

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

sections.opm

```
\_def\_theoutline{}
\_def\_thisoutline#1{\_gdef\_theoutline{#1}}
\_public \thisoutline ;
```

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

sections.opm

```
\_newskip \_savedtitleskip
\_newskip \_savedlastskip
\_def\_abovetitle #1#2{\_savedlastskip=\_lastskip % <whatever-skip>
```

```
   \_ifdim\_lastskip>\_zo \_vskip-\_lastskip \_fi
   \_ifnum\_lastpenalty=11333 \_vskip-\_savedtitleskip \_else #1\_fi
   \_ifdim\_savedlastskip>\_zo \_setbox0=\_vbox{#2\_global\_tmpdim=\_lastskip}%
   \_else \_tmpdim=\_maxdimen \_fi
   \_ifdim\_savedlastskip>\_tmpdim \vskip\_savedlastskip \_else #2\_fi
}
\_def\_belowtitle #1{#1\_global\_savedtitleskip=\_lastskip \_penalty11333 }
```

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

sections.opm

```
\_def\_nbpar{{\_interlinepenalty=10000\_endgraf}}

\_protected\_def\_nl{\_unskip\_hfil\_break}
\_regmacro {\_def\_nl{\_unskip\_space}} {\_def\_nl{\_unskip\_space}} {\_def\_nl{ }}
\_regmacro {\_def\nl{\_unskip\_space}}  {\_def\nl{\_unskip\_space}}  {\_def\nl{ }}

\_public \nbpar \nl ;
```

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

sections.opm

```
\_def \_firstnoindent {\_global\_everypar={\_wipeepar \_setbox7=\_lastbox}}
\_def \_wipeepar {\_global\_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.

sections.opm

```
\_def\_insertmark#1{\_mark{{\_ifnonum\_else\_therefnum\_fi} {\_unexpanded{#1}}}}
```

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.

```
\addto\_chapx {\globaldefs=0 \vfil\break % headline of previous chapter is printed
   \xdef\_runningchap {\_thechapnum: \unexpanded\_ea{\_savedtitle}}}
\def \formathead #1#2{\isempty{#1}\iffalse #1: #2\fi}
\headline = {%
   \ifodd \pageno
       \hfil \ea\formathead\firstmark{}{}%
   \else
       \ifx\_runningchap\_undefined \else Chapter \_runningchap \fi \hfil
   \fi
}
```

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 variant `\_eol:secl` is defined to enable `\bracedparam\secl`⟨*number*⟩ `{`⟨*title-text*⟩`}`.

sections.opm

```
\_def\_secl{\_let\_secle=\_ea \_afterassignment\_secla \_sectionlevel=}
\_sdef{_eol:secl}{\_def\_secle{\_ea\_bracedparam\_ea}\_afterassignment\_secla \_sectionlevel=}
\_def\_secla{\_ifcase\_sectionlevel
   \_or \_secle\_chap \_or \_secle\_sec \_or \_secle\_secc \_else \_ea \_seclp\_fi}
\_eoldef\_seclp#1{\_par \_ifnum\_lastpenalty=0 \_removelastskip\_medskip\_fi
   \_noindent{\_bf #1}\_vadjust{\_nobreak}\_nl\_ignorepars}
\_def\_ignorepars{\_isnextchar\_par{\_ignoresecond\_ignorepars}{}}

\_public \secl ;
```

The `\caption/`⟨*letter*⟩ increases `\_`⟨*letter*⟩`num` counter, 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.

sections.opm

```
\_def\_caption/#1{\_def\_tmpa{#1}\_nospaceafter \_capA}
\_optdef\_capA []{\_trylabel \_incaption}
\_def\_incaption {\_bgroup
   \_ifcsname _\_tmpa num\_endcsname \_ea\_incr \_csname _\_tmpa num\_endcsname
   \_else \_opwarning{Unknown caption /\_tmpa}\_fi
   \_edef\_thecapnum {\_csname _the\_tmpa num\_endcsname}%
   \_edef\_thecaptitle{\_mtext{\_tmpa}}%
   \_ea\_captionformat\_ea{\_tmpa}%
   \_ea\_the \_csname _everycaption\_tmpa\_endcsname
   \_def\_par{\_ifhmode\_nbpar\_egroup\_egroup\_fi}%
   \_ifx\par\_endgraf \_let\par=\_par \_fi
   \_bgroup \_aftergroup\_par
   \_cs{_printcaption\_tmpa}%
}
\_def \_cskip {\_par\_nobreak\_medskip} % space between caption and the object

\_public \caption \cskip ;
```

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

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

The macro `\_printcaptiont` (or `f`) is processed inside group and the `\_par` can be run after this group. If you want to re-define formatting parameters for `\_par`, do this in the macro `\_captionformat`. The `\_captionsep` inserts a separator between auto-generated caption number and the following caption text. Default separator is `\_enspace` but if the caption text starts with dot or colon, then the space is not inserted. A user can write `\caption/t: My table` and “**Table 1.1:** My table” is printed. You can re-define the `\_captionsep` macro if you want to use another separator.

sections.opm

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

If you want to declare a new type of `\caption` with independent counter, you can use following lines, where `\caption/a` for Algorithms are declared:

```
\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...
```

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.

sections.opm

```
\_def\_captionformat#1{\_narrowlastlinecentered\_iindent}
\_def\_narrowlastlinecentered#1{%
   \_leftskip=#1plus1fil
   \_rightskip=#1plus-1fil
   \_parfillskip=0pt plus2fil\_relax
}
```

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

sections.opm

```
\_optdef\_eqmark []{\_trylabel \_ineqmark}
\_def\_ineqmark{\_incr\_dnum
  \_ifinner\_else\_eqno \_fi
  \_wlabel\_thednum \_hbox{\_numprint\_thednum}%
}
\_public \eqmark ;
```

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

sections.opm

```
\_newcount\_counterA \_newcount\_counterB \_newcount\_counterC
\_newcount\_counterD \_newcount\_counterE

\_def\_resetABCDE {\_counterA=0 \_counterB=0 \_counterC=0 \_counterD=0 \_counterE=0 }

\_def \_theAnum {\_othe\_chapnum.\_othe\_secnum.\_the\_counterA}
\_def \_theBnum {\_othe\_chapnum.\_othe\_secnum.\_the\_counterB}
\_def \_theCnum {\_othe\_chapnum.\_othe\_secnum.\_the\_counterC}
\_def \_theDnum {\_othe\_chapnum.\_othe\_secnum.\_the\_counterD}
\_def \_theEnum {\_othe\_chapnum.\_othe\_secnum.\_the\_counterE}

\_def\_numberedpar#1#2{\_ea \_incr \_csname _counter#1\_endcsname
   \_def\_tmpa{#1}\_def\_tmpb{#2}\_numberedparparam}
\_optdef\_numberedparparam[]{%
   \_ea \_printnumberedpar \_csname _the\_tmpa num\_ea\_endcsname\_ea{\_tmpb}}

\_public \numberedpar ;
```

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

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

```
\def\theorem    {\medskip\bgroup\it \numberedpar A{Theorem}}
\def\endtheorem {\par\egroup\medskip}

\theorem Let $M$ be... \endtheorem
```

sections.opm

```
\_def \_printnumberedpar #1#2{\_par
   \_noindent\_wlabel #1%
   {\_bf #2 \_numprint{#1}\_istoksempty\_opt\_iffalse \_space \_the\_opt \_fi.}\_space
   \_ignorespaces
}
```

2.27 Lists, items

lists.opm

```
\_codedecl \begitems {Lists: begitems, enditems <2023-10-20>} % preloaded in format
```

`\_aboveliskip` is used above the list of items,
`\_belowliskip` is used below the list of items,
`\_setlistskip` sets the skip dependent on the current level of items,
`\_listskipab` is `\ilistskipamount` or `\olistskipamount`.

lists.opm

```
\_def\_aboveliskip {\_removelastskip \_penalty-100 \_vskip\_listskipab}
\_def\_belowliskip {\_penalty-200 \_vskip\_listskipab}
\_newskip\_listskipab

\_def\_setlistskip {%
   \_ifnum \_ilevel = 1 \_listskipab = \_olistskipamount \_relax
   \_else  \_listskipab = \_ilistskipamount \_relax
   \_fi}
```

The `\itemnum` is locally reset to zero in each group declared by `\begitems`. So nested lists are numbered independently. Users can set initial value of `\itemnum` to another value after `\beitems` if they want.

Each level of nested lists is indented by the new `\iindent` from left. The default item mark is `\_printitem`.

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

The `\enditems` runs `\_isnextchar\_par{}{\_noindent}` thus the next paragraph is without indentation if there is no empty line between the list and this paragraph (it is similar behavior as after display math).

lists.opm

```
\_newcount\_itemnum  \_itemnum=0
\_newtoks\_printitem

\_def\_begitems{\_par
  \_bgroup
  \_advance \_ilevel by1
  \_setlistskip
  \_ifnum\_lastpenalty<10000 \_aboveliskip \_fi
  \_itemnum=0 \_adef*{\_relax\_ifmmode*\_else\_ea\_startitem\_fi}
  \_advance\_leftskip by\_iindent
  \_printitem=\_defaultitem
  \_the\_everylist \_relax
}
\_def\_enditems{\_par\_belowliskip\_egroup \_isnextchar\_par{}{\_noindent}}

\_def\_startitem{\_par \_ifnum\_itemnum>0 \_vskip\_itemskipamount \_fi
   \_advance\_itemnum by1
   \_the\_everyitem \_noindent\_llap{\_the\_printitem}\_ignorespaces
}
\_public \begitems \enditems \itemnum ;
```

`\novspaces` sets `\_listskipab` and `\itemskipamount` to 0pt. Moreover, it deactivates `\_setlistskip` (for inner lists).

lists.opm

```
\_def\_novspaces {\_removelastskip
   \_listskipab=\_zoskip \_itemskipamount=\_zoskip \_let\_setlistskip=\_relax}
\_public \novspaces ;
```

Various item marks are saved in `\_item:`⟨*letter*⟩ macros. You can re-define then or define more such macros. The `\style` ⟨*letter*⟩ does `\_printitem={\_item:`⟨*letter*⟩`}`. More exactly: `\begitems` does `\_printitem=\defaultitem` first, then `\style` ⟨*letter*⟩ does `\_printitem={\_item:`⟨*letter*⟩`}` when it is used and finally, `\_startitem` alias `*` uses `\_printitem`.

lists.opm

```
\_def\_style#1{%
   \_ifcsname _item:#1\_endcsname \_printitem=\_ea{\_csname _item:#1\_endcsname}%
   \_else \_printitem=\_defaultitem \_fi
}
\_sdef{_item:o}{\_raise.4ex\_hbox{$\_scriptscriptstyle\_bullet$} }
\_sdef{_item:-}{- }
\_sdef{_item:n}{\_the\_itemnum. }
\_sdef{_item:N}{\_the\_itemnum) }
\_sdef{_item:i}{(\_romannumeral\_itemnum) }
\_sdef{_item:I}{\_uppercase\_ea{\_romannumeral\_itemnum}\_kern.5em}
\_sdef{_item:a}{\_athe\_itemnum) }
\_sdef{_item:A}{\_uppercase\_ea{\_athe\_itemnum}) }
\_sdef{_item:x}{\_raise.3ex\_fullrectangle{.6ex}\_kern.4em}
\_sdef{_item:X}{\_raise.2ex\_fullrectangle{1ex}\_kern.5em}
\_sdef{_item:d}{\_aftergroup\_dword}
\_def\_dword#1#2{{\_bf #2 }\_ignorespaces} % #1 is \_ignorespaces from \_startitem
```

`\_athe{`⟨*num*⟩`}` returns the ⟨*num*⟩s lowercase letter from the alphabet.

`\_fullrectangle{`⟨*dimen*⟩`}` prints full rectangle with given ⟨*dimen*⟩.

lists.opm

```
\_def\_fullrectangle#1{\_hbox{\_vrule height#1 width#1}}

\_def\_athe#1{\_ifcase#1?\_or a\_or b\_or c\_or d\_or e\_or f\_or g\_or h\_or
   i\_or j\_or k\_or l\_or m\_or n\_or o\_or p\_or q\_or r\_or s\_or t\_or
   u\_or v\_or w\_or x\_or y\_or z\_else ?\_fi
}
\_public \style ;
```

The `\begblock` macro selects fonts from footnotes `\_fnset` and opens new indentation in a group. `\endblock` closes the group. This is implemented as an counterpart of Markdown's Blockquotes. Redefine these macros if you want to declare different design. The OpTEX trick 0031 shows how to create blocks with grey background splittable to more pages.

lists.opm

```
\_def\_begblock{\_bgroup\_fnset \_medskip \_advance\_leftskip by\_iindent \_firstnoindent}
\_def\_endblock{\_par\_medskip\_egroup\_isnextchar\_par{}{\_noindent}}

\_public \begblock \endblock ;
```

2.28 Verbatim, listings

2.28.1 Inline and “display” verbatim

verbatim.opm

```
\_codedecl \begtt {Verbatim <2024-11-05>} % preloaded in format
```

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

verbatim.opm

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

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

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

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

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

verbatim.opm

```
\_def\_code#1{\_unexpanded\_ea{\_csname _code \_endcsname{#1}}}
\_protected\_sdef{_code }#1{{\_escapechar=-1 \_ttfont \_the\_everyintt \_relax
   \_ea\_printinverbatim\_ea{\_detokenize{#1}}}}
\_def\_printinverbatim#1{\_leavevmode\_hbox{#1}}

\_regmacro {}{}{\_let\code=\_detokenize \_let\_code=\_detokenize}
\_public \code ;
```

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

verbatim.opm

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

```
\_def "{\_begingroup \_setverb ... \_readverb}
\_def \_readverb #1"{\_printinverbatim{#1}\_endgroup}
```

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

`verbatim.opm`

```
\_def\_verbchar#1{%
   \_ifx\_savedttchar\_undefined\_else \_catcode\_savedttchar=\_savedttcharc \_fi
   \_chardef\_savedttchar=`#1
   \_chardef\_savedttcharc=\_catcode`#1
   \_adef{#1}{\_begingroup \_setverb \_adef{ }{\_dsp}\_ttfont \_the\_everyintt\_relax \_readverb}%
   \_def\_readverb ##1#1{\_printinverbatim{##1}\_endgroup}%
}
\_let \_activettchar=\_verbchar % for backward compatibility
\_public \verbchar \activettchar ;
```

`\begtt` is defined only as public. We don't need a private `\_begtt` variant. This macro opens a group and sets `%` as an active character (temporary). This will allow it to be used as the comment character at the same line after `\begtt`. Then `\_begtti` is run. It is defined by `\eoldef`, so users can put a parameter at the same line where `\begtt` is. This `#1` parameter is used after `\everytt` parameters settings, so users can change them locally.

A user can write `\begtt \ttline=0` and we want to interpret this setting locally too. This is the reason of creating the `\_restorettline` macro before the argument `#1` of `\_begtti` is processed and using the trick `\ifnum\ttline=\maxdimen` after this processing. See the code below.

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

The `\_startverb` puts the scanned data to `\_prepareverbdata`. It sets the data to `\_tmpb` without changes by default, but you should re-define it in order to do special changes if you want. (For example, `\hisyntax` redefines this macro.) The scanned data have `^^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.

`verbatim.opm`

```
\_def\begtt{\_par \_begingroup \_adef##1\_relax{\_relax}\_begtti}
\_eoldef \_begtti#1{\_wipeepar \_setxhsize
   \_vskip\_parskip \_ttskip
   \_setverb
   \_adef{ }{\_dsp}\_adef^I{\t}\_parindent=\_ttindent \_parskip=0pt
   \_def\t{\_hskip \_dimexpr\_tabspaces em/2\_relax}%
   \_protrudechars=0 % disable protrusion
   \_xdef\_restorettline{\_global\_ttline=\_the\_ttline\_relax}\_ttline=\_maxdimen
   \_the\_everytt \_relax #1\_relax
   \_ifnum\_ttline=\_maxdimen \_restorettline \_else \_aftergroup\_restorettline \_fi
   \_ifnum\_ttline<0 \_let\_printverblinenum=\_relax \_else \_initverblinenum \_fi
   \_def\_testcommentchars##1\_iftrue{\_iffalse}\_let\_hicomments=\_relax
   \_ttfont \_savemathsb \_endlinechar=`^^J
   \_startverb
}
\_ea\_def\_ea\_startverb \_ea#\_ea1\_csstringendtt#2^^J{%
   \_prepareverbdata\_tmpb{#1^^J}%
   \_ea\_printverb \_tmpb\_fin
   \_par \_restoremathsb
   \_endgroup \_ttskip
   \_isnextchar\_par{}{\_noindent}%
}
\_def\_prepareverbdata#1#2{\_def#1{#2}}
```

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 `\_printverblinenum` is used here: it does nothing when `\_ttline<0` else it prints the line number using `\_llap`.

`\_putttpenalty` puts `\_ttpenalty` before second and next lines, but not before first line in each `\begtt...\endtt` environment.

The `\_ttline` is increased here in the `\_printverb` macro because of comments-blocks: the `\_prinverbline` is not processed in comments-blocks but we need to count the `\_ttline`.

verbatim.opm

```
\_def\_printverb #1^^J#2{%
   \_ifx\_printverblinenum\_relax \_else \_incr\_ttline \_fi
   \_testcommentchars #1\_relax\_relax\_relax
   \_iftrue
      \_ifx\_fin#2\_printcomments\_fi
   \_else
      \_ifx\_vcomments\_empty\_else  \_printcomments \_def\_vcomments{}\_fi
      \_ifx\_fin#2%
         \_bgroup \_adef{ }{}\_def\t{}% if the last line is empty, we don't print it
         \_ifcat&#1&\_egroup \_ifx\_printverblinenum\_relax \_else \_decr\_ttline \_fi
         \_else\_egroup \_printverbline{#1}\_fi
      \_else
         \_printverbline{#1}%
      \_fi
   \_fi
   \_unless\_ifx\_fin#2\_afterfi{\_printverb#2}\_fi
}
\_def\_printverbline#1{\_putttpenalty \_indent \_printverblinenum \_kern\_ttshift #1\_par}
\_def\_initverblinenum{\_tenrm \_thefontscale[700]\_ea\_let\_ea\_sevenrm\_the\_font}
\_def\_printverblinenum{\_llap{\_sevenrm \_the\_ttline\_kern.9em}}
\_def\_putttpenalty{\_def\_putttpenalty{\_penalty\_ttpenalty}}
```

Macro `\verbinput` uses a file read previously or opens the given file. Then it runs the parameter scanning by `\_viscanparameter` and `\_viscanminus`. Finally the `\_doverbinput` is run. At the beginning of `\_doverbinput`, we have `\_viline`= number of lines already read using previous `\verbinput`, `\_vinolines`= the number of lines we need to skip and `\_vidolnes`= 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.

verbatim.opm

```
\_def\_verbinput #1(#2) #3 {\_par \_def\_tmpa{#3}%
   \_def\_tmpb{#1}%  cmds used in local group
   \_ifx\_vifilename\_tmpa \_else
      \_openin\_vifile={#3}%
      \_global\_viline=0 \_glet\_vifilename=\_tmpa
      \_ifeof\_vifile
         \_opwarning{\_string\verbinput: file "#3" unable to read}
         \_ea\_ea\_ea\_skiptorelax
      \_fi
   \_fi
   \_viscanparameter #2+\_relax
}
\_def\_skiptorelax#1\_relax{}

\_def \_viscanparameter #1+#2\_relax{%
   \_if$#2$\_viscanminus(#1)\_else \_viscanplus(#1+#2)\_fi
}
\_def\_viscanplus(#1+#2+){%
   \_if$#1$\_tmpnum=\_viline
   \_else \_ifnum#1<0 \_tmpnum=\_viline \_advance\_tmpnum by-#1
       \_else \_tmpnum=#1
             \_advance\_tmpnum by-1
             \_ifnum\_tmpnum<0 \_tmpnum=0 \_fi % (0+13) = (1+13)
   \_fi \_fi
   \_edef\_vinolines{\_the\_tmpnum}%
   \_if$#2$\_def\_vidolines{0}\_else\_edef\_vidolines{#2}\_fi
   \_doverbinput
}
\_def\_viscanminus(#1-#2){%
   \_if$#1$\_tmpnum=0
      \_else \_tmpnum=#1 \_advance\_tmpnum by-1 \_fi
   \_ifnum\_tmpnum<0 \_tmpnum=0 \_fi  % (0-13) = (1-13)
   \_edef\_vinolines{\_the\_tmpnum}%
   \_if$#2$\_tmpnum=0
      \_else \_tmpnum=#2 \_advance\_tmpnum by-\_vinolines \_fi
   \_edef\_vidolines{\_the\_tmpnum}%
   \_doverbinput
}
\_def\_doverbinput{%
   \_tmpnum=\_vinolines
   \_advance\_tmpnum by-\_viline
   \_ifnum\_tmpnum<0
      \_openin\_vifile={\_vifilename}%
      \_global\_viline=0
   \_else
      \_edef\_vinolines{\_the\_tmpnum}%
   \_fi
   \_vskip\_parskip \_ttskip \_wipeepar \_setxhsize
   \_begingroup
   \_setverb \_adef{ }{\_dsp}\_adef^I{\t}\_parindent=\_ttindent \_parskip=0pt
   \_def\t{\_hskip \_dimexpr\_tabspaces em/2\_relax}%
   \_protrudechars=0 % disable protrusion
   \_xdef\_restorettline{\_global\_ttline=\_the\_ttline\_relax}\_ttline=\_maxdimen
   \_the\_everytt\_relax \_tmpb\_relax
   \_ifnum\_ttline=\_maxdimen \_restorettline \_else \_aftergroup\_restorettline \_fi
   \_ifnum\_ttline<-1 \_let\_printverblinenum=\_relax \_else \_initverblinenum \_fi
   \_ttfont \_savemathsb \_endlinechar=`^^J \_tmpnum=0
   \_loop \_ifeof\_vifile \_tmpnum=\_vinolines\_space \_fi
         \_ifnum\_tmpnum<\_vinolines\_space
         \_vireadline \_advance\_tmpnum by1 \_repeat      %% skip lines
   \_edef\_ttlinesave{\_global\_ttline=\_the\_ttline}%
   \_ifnum\_ttline=-1 \_ttline=\_viline \_else \_let\_ttlinesave=\_relax \_fi
   \_tmpnum=0 \_def\_tmpb{}%
   \_ifnum\_vidolines=0 \_tmpnum=-1 \_fi
   \_ifeof\_vifile \_tmpnum=\_vidolines\_space \_fi
   \_loop \_ifnum\_tmpnum<\_vidolines\_space
            \_vireadline
            \_ifnum\_vidolines=0 \_else\_advance\_tmpnum by1 \_fi
            \_ifeof\_vifile \_tmpnum=\_vidolines\_space \_else \_visaveline \_fi %% save line
            \_repeat
   \_ea\_prepareverbdata \_ea \_tmpb\_ea{\_tmpb^^J}%
   \_catcode`=10 \_catcode`=9 % used in \commentchars comments
   \_ea\_printverb \_tmpb\_fin
   \_ttlinesave
   \_par \_restoremathsb
   \_endgroup
   \_ttskip
   \_isnextchar\_par{}{\_noindent}%
}
\_def\_vireadline{\_read\_vifile to \_tmp \_incr\_viline }
\_def\_visaveline{\_ea\_addto\_ea\_tmpb\_ea{\_tmp}}

\_public \verbinput ;
```

`\_savemathsb`, `\_restoremathsb` pair is used in `\begtt`...`\endtt` or in `\verbinput` to temporary suppress the `\mathsbon` because we don't need to print `\int _a` in verbatim mode if `\int_a` is really written. The `\_restoremathsb` is defined locally as `\mathsbon` only if it is needed.

verbatim.opm

```
\_def\_savemathsb{\_ifmathsb \_mathsboff \_def\_restoremathsb{\_mathsbon}\_fi}
\_def\_restoremathsb{}
```

If the language of your code printed by `\verbinput` supports the format of comments started by two characters from the beginning of the line then you can set these characters by `\commentchars`⟨*first*⟩⟨*second*⟩.

Such comments are printed in the non-verbatim mode without these two characters and they look like the verbatim printing is interrupted at the places where such comments are. See the section 2.39 for good illustration. The file `optex.lua` is read by a single command `\verbinput (4-) optex.lua` here and the `\commentchars --` was set before it.

If you need to set a special character by `\commentchars` then you must to set the catcode to 12 (and space to 13). Examples:

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

There is one limitation when TeX interprets the comments declared by `\commentchars`. Each block of comments is accumulated to one line and then it is re-interpreted by TeX. So, the ends of lines in the comments block are lost. You cannot use macros which need to scan end of lines, for example `\begtt...\endtt` inside the comments. The character `%` is ignored in comments but you can use `\%` for printing or `%` alone for de-activating `\_endpar` from empty comment lines.

Implementation: The `\commentchars`⟨*first*⟩⟨*second*⟩ redefines the `\_testcommentchars` used in `\_printverb` in order to it removes the following `\_iftrue` and returns `\_iftrue` or `\_iffalse` depending on the fact that the comment characters are or aren't present at the beginning of tested line. If it is true (`\ifnum` expands to `\ifnum 10>0`) then the rest of the line is added to the `\_vcomments` macro.

The `\_hicomments` is `\relax` by default but it is redefined by `\commentchars` in order to keep no-colorized comments if we need to use feature from `\commentchars`.

The accumulated comments are printed whenever the non-comment line occurs. This is done by `\_printcomments` macro. You can re-define it, but the main idea must be kept: it is printed in the group, `\_reloding \_rm` initializes normal font, `\catcodetable0` returns to normal catcode table used before `\verbinput` is started, and the text accumulated in `\_vcomments` must be printed by `\_scantextokens` primitive.

verbatim.opm

```
\_def\_vcomments{}
\_let\_hicomments=\_relax

\_def\_commentchars#1#2{%
   \_def\_testcommentchars ##1##2##3\_relax ##4\_iftrue{\_ifnum % not closed in this macro
      \_ifx #1##1\_ifx#2##21\_fi\_fi 0>0
      \_ifx\_relax##3\_relax \_addto\_vcomments{\_endgraf}% empty comment=\enfgraf
      \_else \_addto\_vcomments{##3 }\_fi}%
   \_def\_hicomments{\_replfromto{\b\n#1#2}{^^J}{\w{#1#2####1}^^J}}% used in \hisyntax
}
\_def\_testcommentchars #1\_iftrue{\_iffalse} % default value of \_testcommentchar
\_def\_printcomments{\_ttskip
   {\_catcodetable0 \_rm \_everypar={}%
    \_noindent \_ignorespaces \_scantextokens\_ea{\_vcomments}\_par}%
   \_ttskip
}
\_public \commentchars ;
```

The `\visiblesp` sets spaces as visible characters ␣. It redefines the `\_dsp`, so it is useful for verbatim modes only.

The `\_dsp` is equivalent to `\␣` primitive. It is used in all verbatim environments: spaces are active and defined as `\_dsp` here.

verbatim.opm

```
\_def \_visiblesp{\_ifx\_initunifonts\_relax \_def\_dsp{\_char9251 }%
                 \_else \_def\_dsp{\_char32 }\_fi}
\_let\_dsp= % primitive "direct space"

\_public \visiblesp ;
```

2.28.2 Listings with syntax highlighting

The user can write

```
\begtt \hisyntax{C}
...
\endtt
```

to colorize the code using C syntax. The user can also write `\everytt={\hisyntax{C}}` to have all verbatim listings colorized.

`\hisyntax{`⟨*name*⟩`}` reads the file `hisyntax-`⟨*name*⟩`.opm` where the colorization is declared. The parameter ⟨*name*⟩ is case insensitive and the file name must include it in lowercase letters. For example, the file `hisyntax-c.opm` looks like this:

hisyntax-c.opm

```
\_codedecl \_hisyntaxc {Syntax highlighting for C sources <2023-03-02>}

\_newtoks \_hisyntaxc  \_newtoks \_hicolorsc

\_global\_hicolorsc={%     colors for C language
   \_hicolor K \Red      % Keywords
   \_hicolor S \Magenta  % Strings
   \_hicolor C \Green    % Comments
   \_hicolor N \Cyan     % Numbers
   \_hicolor P \Blue     % Preprocessor
   \_hicolor O \Blue     % Non-letters
}
\_global\_hisyntaxc={%
   \_the\_hicolorsc
   \_let\c=\_relax \_let\e=\_relax \_let\o=\_relax
   \_replfromto {/*}{*/}          {\x C{/*#1*/}}%   /*...*/
   \_replfromto {//}{^^J}         {\z C{//#1}^^J}%  //...
   \_replfromto {\_string#}{^^J} {\z P{#1}^^J}%  #include ...
   \_replthis   {\_string}     {{\_string}}%    protected inside strings
   \_replfromto {"}{"}            {\x S{"#1"}}%     "..."
   %
   \_edef\_tmpa {()\_string{\_string}+-*/=[]<>,:;\_pcent\_string&\_string^|!?}% non-letters
   \_ea \_foreach \_tmpa
      \_do {\_replthis{#1}{\n\o#1\n}}
   \_foreach                                                               % keywords
      {alignas}{alignof}{auto}{bool}{break}{case}{char}{const}%
      {constexpr}{continue}{default}{do}{double}{else}{enum}{extern}%
      {false}{float}{for}{goto}{if}{inline}{int}{long}{nullptr}%
      {register}{restrict}{return}{short}{signed}{sizeof}{static}%
      {static_assert}{struct}{switch}{thread_local}{true}{typedef}%
      {typeof}{typeof_unqual}{union}{unsigned}{void}{volatile}{while}%
      {_Alignas}{_Alignof}{_Atomic}{_BitInt}{_Bool}{_Complex}%
      {_Decimal128}{_Decimal32}{_Decimal64}{_Generic}{_Imaginary}%
      {_Noreturn}{_Static_assert}{_Thread_local}
      \_do {\_replthis{\n#1\n}{\z K{#1}}}
   \_replthis{.}{\n.\n}                                                    % numbers
   \_foreach 0123456789
      \_do {\_replfromto{\n#1}{\n}{\c#1##1\e}}
   \_replthis{\e.\c}{.}
   \_replthis{\e.\n}{.\e}
   \_replthis{\n.\c}{\c.}
   \_replthis{e\e\o+\c}{e+}\_replthis{e\e\o-\c}{e-}
   \_replthis{E\e\o+\c}{E+}\_replthis{E\e\o-\c}{E-}
   \_def\o#1{\z O{#1}}
   \_def\c#1\e{\z N{#1}}
}
```

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

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

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

Notes for hi-syntax macro writers

The file `hisyntax-`⟨*name*⟩`.opm` is read only once and in a TEX group. If there are definitions then they must be declared as global.

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

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

The pre-processing phase does:

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

Special macros are working only in a group when processing the verbatim text.

- `\n` represents nothing but it should be used as a boundary of words as mentioned above.
- `\t` represents a tabulator. It is prepared as `\n\t\n` because it can be at the boundary word boundary.
- `\x` ⟨*letter*⟩`{`⟨*text*⟩`}` can be used as replacing text. Consider the example

  ```
  \replfromto{/*}{*/}{\x C{/*#1*/}}
  ```

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

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

  ```
  \x C{⟨text1⟩^^J⟨text2⟩^^J⟨text3⟩}
  ```

  is replaced by

  ```
  \z C{⟨text1⟩}^^J\z C{⟨text2⟩}^^J\z C{⟨text3⟩}
  ```

  `\z` ⟨*letter*⟩`{`⟨*text*⟩`}` is expanded to `\_z:`⟨*letter*⟩`{`⟨*text*⟩`}` and if `\hicolor` ⟨*letter*⟩ ⟨*color*⟩ is declared then `\_z:`⟨*letter*⟩`{`⟨*text*⟩`}` expands to `{`⟨*color*⟩⟨*text*⟩`}`. So, required color is activated for each line separately (e.g. for C comments spanning multiple lines).
- `\y` `{`⟨*text*⟩`}` is replaced by `\`⟨*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.

hi-syntax.opm

```
\_codedecl \hisyntax {Syntax highlighting of verbatim listings <2022-04-04>} % preloaded in format
```

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

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

`\replfromto` continues by finding next ⟨*from*⟩, then, next ⟨*to*⟩ repeatedly over the whole verbatim text. If the verbatim text ends with opening ⟨*from*⟩ but has no closing ⟨*to*⟩, then ⟨*to*⟩ is appended to the verbatim text 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.

hi-syntax.opm

```
\_def\_replfromto #1#2{\_edef\_tmpa{{#1}{#2}}\_ea\_replfromtoE\_tmpa}
\_def\_replfromtoE#1#2#3{% #1=from #2=to #3=replacement
   \_def\_replfrom##1#1##2{\_addto\_tmpb{##1}%
      \_ifx\_fin##2\_ea\_replstop \_else \_afterfi{\_replto##2}\_fi}%
   \_def\_replto##1#2##2{%
      \_ifx\_fin##2\_afterfi{\_replfin##1}\_else
        \_addto\_tmpb{#3}%
        \_afterfi{\_replfrom##2}\_fi}%
   \_def\_replfin##1#1\_fin{\_addto\_tmpb{#3}\_replstop}%
   \_edef\_tmpb{\_ea}\_ea\_replfrom\_tmpb#1\_fin#2\_fin\_fin\_relax
}
\_def\_replstop#1\_fin\_relax{}
\_def\_finrepl{}
```

The `\replthis` `{`⟨*pattern*⟩`}{`⟨*replacement*⟩`}` replaces each ⟨*pattern*⟩ by ⟨*replacement*⟩. Both parameters of `\replthis` are expanded first.

hi-syntax.opm

```
\_def\_replthis#1#2{\_edef\_tmpa{{#1}{#2}}\_ea\_replstring\_ea\_tmpb \_tmpa}

\_public \replfromto \replthis ;
```

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.

hi-syntax.opm

```
\_def\_xscan#1#2^^J#3{\_ifx\_fin#3 \_ea\_xscanR\_fi
   \z{#1}{#2}%
   \_ifx^#3\_else ^^J\_afterfi{\_xscan{#1}#3}\_fi}
\_def\_xscanR#1\_fi#2^{^^J}
```

The `\hicolor` ⟨*letter*⟩ ⟨*color*⟩ defines `\_z:`⟨*letter*⟩`{`⟨*text*⟩`}` as `{`⟨*color*⟩⟨*text*⟩`}`. It should be used in the context of `\x` ⟨*letter*⟩`{`⟨*text*⟩`}` macros.

hi-syntax.opm

```
\_def\_hicolor #1#2{\_sdef{_z:#1}##1{{#2##1}}}
```

`\hisyntax{`⟨*name*⟩`}` re-defines default `\_prepareverbdata`⟨*macro*⟩⟨*verbtext*⟩, but in order to do it does more things: It saves ⟨*verbtext*⟩ to `\_tmpb`, appends `\n` around spaces and `^^J` characters in pre-processing phase, opens `hisyntax-`⟨*name*⟩`.opm` file if `\_hisyntax`⟨*name*⟩ is not defined. Then `\_the\_hisyntax`⟨*name*⟩ is processed. Finally, the post-processing phase is realized by setting appropriate values to the `\x` and `\y` macros and doing `\_edef\_tmpb{\_tmpb}`.

hi-syntax.opm

```
\_def\_hisyntax#1{\_def\_prepareverbdata##1##2{%
   \_let\n=\_relax \_let\b=\_relax \_def\t{\n\_noexpand\t\n}\_let\_start=\_relax
   \_adef{ }{\n\_noexpand\n}\_edef\_tmpb{\_start^^J##2\_fin}%
   \_replthis{^^J}{\n^^J\b\n}\_replthis{\b\n\_fin}{\_fin}%
   \_let\x=\_relax  \_let\y=\_relax \_let\z=\_relax \_let\t=\_relax
   \_hicomments % keeps comments declared by \commentchars
   \_endlinechar=`^^M
   \_lowercase{\_def\_tmpa{#1}}%
   \_ifcsname _hialias:\_tmpa\_endcsname \_edef\_tmpa{\_cs{_hialias:\_tmpa}}\_fi
   \_ifx\_tmpa\_empty \_else
      \_unless \_ifcsname _hisyntax\_tmpa\_endcsname
          \_isfile{hisyntax-\_tmpa.opm}\_iftrue \_opinput {hisyntax-\_tmpa.opm} \_fi\_fi
      \_ifcsname _hisyntax\_tmpa\_endcsname
          \_ifcsname hicolors\_tmpa\_endcsname
              \_cs{_hicolors\_tmpa}=\_cs{hicolors\_tmpa}%
          \_fi
          \_ea\_the \_csname _hisyntax\_tmpa\_endcsname % \_the\_hisyntax<name>
          \_the\_hicolors  % colors which have precedece
      \_else\_opwarning{Syntax "\_tmpa" undeclared (no file hisyntax-\_tmpa.opm)}
   \_fi\_fi
   \_replthis{\_start\n^^J}{}\_replthis{^^J\_fin}{^^J}%
   \_def\n{}\_def\b{}\_adef{ }{\_dsp}%
   \_bgroup \_lccode`=`\_lowercase{\_egroup\_def{\_noexpand~}}%
   \_def\w####1{####1}\_def\x####1####2{\_xscan{####1}####2^^J^}%
   \_def\y####1{\_ea \_noexpand \_csname ####1\_endcsname}%
   \_edef\_tmpb{\_tmpb}%
   \_def\z####1{\_cs{_z:####1}}%
   \_def\t{\_hskip \_dimexpr\_tabspaces em/2\_relax}%
   \_localcolor
}}
\_public \hisyntax \hicolor ;
```

Aliases for languages can be declared like this. When `\hisyntax{xml}` is used then this is the same as `\hisyntax{html}`.

hi-syntax.opm

```
\_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{⟨picture⟩}}

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

This code downloads the picture data to the PFD output only once (when `\setbox` is processed). Each usage of `\copy\mypic` puts only a pointer to the picture data in the PDF.

If you want to copy the same picture in different sizes, then choose a “basic size” used in `\setbox` and all different sizes can be realized by the `\transformbox{`⟨*transformation*⟩`}{\copy\mypic}`.

graphics.opm

```
\_codedecl \inspic {Graphics <2023-03-16>} % preloaded in format
```

`\inspic` accepts old syntax `\inspic` ⟨*filename*⟩⟨*space*⟩ or new syntax `\inspic{`⟨*filename*⟩`}`. So, we need to define two auxiliary macros `\_inspicA` and `\_inspicB`.

All `\inspic` macros are surrounded in `\hbox` in order user can write `\moveright\inspic ...` or something similar.

graphics.opm

```
\_def\_inspic{\_hbox\_bgroup\_isnextchar\_bgroup\_inspicB\_inspicA}
\_def\_inspicA #1 {\_inspicB {#1}}
\_def\_inspicB #1{%
   \_pdfximage \_ifdim\_picwidth=\_zo  \_else width\_picwidth\_fi
               \_ifdim\_picheight=\_zo \_else height\_picheight\_fi
               \_the\_picparams {\_the\_picdir#1}%
   \_pdfrefximage\_pdflastximage\_egroup}

\_public \inspic ;
```

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.

graphics.opm

```
\_def\_inkinspic{\_hbox\_bgroup\_isnextchar\_bgroup\_inkinspicB\_inkinspicA}
\_def\_inkinspicA #1 {\_inkinspicB {#1}}
\_def\_inkinspicB #1{%
  \_ifdim\_picwidth=0pt \_setbox0=\_hbox{\_inspic{#1}}\_picwidth=\_wd0 \_fi
  \_tmptoks={#1}%
  \_the\_inkdefs
  \_opinput {\_the\_picdir #1_tex}% file with labels
  \_egroup}

\_newtoks\_tmptoks
\_newtoks\_inkdefs  \_inkdefs={%
  \_def\makeatletter#1\makeatother{}%
  \_def\includegraphics[#1]#2{\_inkscanpage#1,page=,\_fin \_inspic{\_the\_tmptoks}\_hss}%
  \_def\_inkscanpage#1page=#2,#3\_fin{\_ifx,#2,\_else\_picparams{page#2}\_fi}%
  \_def\put(#1,#2)#3{\_nointerlineskip\_vbox to\_zo{\_vss\_hbox to\_zo{\_kern#1\_picwidth
      \_pdfsave\_hbox to\_zo{#3}\_pdfrestore\_hss}\_kern#2\_picwidth}}%
  \_def\begin#1{\_csname _begin#1\_endcsname}%
  \_def\_beginpicture(#1,#2){\_vbox\_bgroup
      \_hbox to\_picwidth{}\_kern#2\_picwidth \_def\end##1{\_egroup}}%
  \_def\_begintabular[#1]#2#3\end#4{%
      \_vtop{\_def{\_cr}\_tabiteml{}\_tabitemr{}\_table{#2}{#3}}}%
  \_def\color[#1]#2{\_scancolor #2,}%
  \_def\_scancolor#1,#2,#3,{\_pdfliteral{#1 #2 #3 rg}}%
  \_def\makebox(#1)[#2]#3{\_hbox to\_zo{\_csname _mbx:#2\_endcsname{#3}}}%
  \_sdef{_mbx:lb}#1{#1\_hss}\_sdef{_mbx:rb}#1{\_hss#1}\_sdef{_mbx:b}#1{\_hss#1\_hss}%
  \_sdef{_mbx:lt}#1{#1\_hss}\_sdef{_mbx:rt}#1{\_hss#1}\_sdef{_mbx:t}#1{\_hss#1\_hss}%
  \_def\rotatebox#1#2{\_pdfrotate{#1}#2}%
  \_def\lineheight#1{}%
  \_def\setlength#1#2{}%
  \_def\transparent#1{\_transparency\_exprA[0]{(1-#1)*255} }%
  % Inkscape may generate \textbf{\textit{\textsc{TEXT}}}
  \_def\textbf#1{\_begingroup\_let\_it\_bi\_bf #1\_endgroup}%
  \_def\textit#1{\_begingroup\_it #1\_endgroup}%
  \_def\textsl#1{\_begingroup\_trycs{slant}{}\_it #1\_endgroup}%
}
\_public \inkinspic ;
```

`\pdfscale`{⟨*x-scale*⟩}{⟨*y-scale*⟩} and `\pdfrotate`{⟨*degrees*⟩} macros are implemented by `\pdfsetmatrix` primitive. We need to know the values of sin, cos function in the `\pdfrotate`. We use Lua code for this.

graphics.opm

```
\_def\_pdfscale#1#2{\_pdfsetmatrix{#1 0 0 #2}}

\_def\_gonfunc#1#2{%
   \_directlua{tex.print(string.format('\_pcent.4f',math.#1(3.14159265*(#2)/180)))}%
}
\_def\_sin{\_gonfunc{sin}}
\_def\_cos{\_gonfunc{cos}}

\_def\_pdfrotate#1{\_pdfsetmatrix{\_cos{#1} \_sin{#1} \_sin{(#1)-180} \_cos{#1}}}

\_public \pdfscale \pdfrotate ;
```

The `\transformbox`{⟨*transformation*⟩}{⟨*text*⟩} is copied from OPmac trick 0046.
The `\rotbox`{⟨*degrees*⟩}{⟨*text*⟩} is a combination of `\rotsimple` from OPmac trick 0101 and the `\transformbox`. Note, that `\rotbox{-90}` puts the rotated text to the height of the outer box (depth is zero) because code from `\rotsimple` is processed. But `\rotbox{-90.0}` puts the rotated text to the depth of the outer box (height is zero) because `\transformbox` is processed.

graphics.opm

```
\_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
      \_ea\_ignorept\_the\_vvalX\_space \_ea\_ignorept\_the\_vvalY\_space}%
}
\_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
   \_preptransform{#1}%
   \_kern-\_newLt \_vrule height\_newHt depth\_newDp width\_zo
   \_setbox0=\_hbox{\_box0}\_ht0=\_zo \_dp0=\_zo
   \_pdfsave#1\_rlap{\_box0}\_pdfrestore \_kern\_newRt}%
}
\_def\_preptransform #1{\_def\_currmatrix{1 0 0 1 }%
   \_def\_pdfsetmatrix##1{\_edef\_tmpb{##1 }\_ea\_multiplyMxM \_tmpb\_unskip}%
   \_let\pdfsetmatrix=\_pdfsetmatrix #1%
   \_setnewHtDp 0pt  \_ht0  \_setnewHtDp 0pt  -\_dp0
   \_setnewHtDp \_wd0 \_ht0  \_setnewHtDp \_wd0 -\_dp0
   \_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
}

```

```
\_def\_rotbox#1#2{%
   \_isequal{90}{#1}\_iftrue \_rotboxA{#1}{\_kern\_ht0 \_tmpdim=\_dp0}{\_vfill}{#2}%
   \_else \_isequal{-90}{#1}\_iftrue \_rotboxA{#1}{\_kern\_dp0 \_tmpdim=\_ht0}{}{#2}%
   \_else \_transformbox{\_pdfrotate{#1}}{#2}%
   \_fi \_fi
}
\_def\_rotboxA #1#2#3#4{\_hbox{\_setbox0=\_hbox{{#4}}#2%
   \_vbox to\_wd0{#3\_wd0=\_zo \_dp0=\_zo \_ht0=\_zo
                  \_pdfsave\_pdfrotate{#1}\_box0\_pdfrestore\vfil}%
   \_kern\_tmpdim
}}
\_public \transformbox \rotbox ;
```

`\_scantwodimens` scans two objects with the syntactic rule ⟨*dimen*⟩ and returns `{`⟨*number*⟩`}{`⟨*number*⟩`}` in `sp` unit.
`\puttext` ⟨*right*⟩ ⟨*up*⟩`{`⟨*text*⟩`}` puts the ⟨*text*⟩ to desired place: From current point moves ⟨*down*⟩ and ⟨*right*⟩, puts the ⟨*text*⟩ and returns back. The current point is unchanged after this macro ends.
`\putpic` ⟨*right*⟩ ⟨*up*⟩ ⟨*width*⟩ ⟨*height*⟩ `{`⟨*image-file*⟩`}` does `\puttext` with the image scaled to desired ⟨*width*⟩ and ⟨*height*⟩. If ⟨*with*⟩ or ⟨*height*⟩ is zero, natural dimension is used. The `\nospec` is a shortcut to such a natural dimension.
`\backgroundpic{`⟨*image-file*⟩`}` puts the image to the background of each page. It is used in the `\slides` style, for example.

graphics.opm

```
\_def\_scantwodimens{%
   \_directlua{tex.print(string.format('{\_pcent d}{\_pcent d}',
               token.scan_dimen(),token.scan_dimen()))}%
}

\_def\_puttext{\_ea\_ea\_ea\_puttextA\_scantwodimens}
\_long\_def\_puttextA#1#2#3{{\_setbox0=\_hbox{{#3}}\_dimen1=#1sp \_dimen2=#2sp \_puttextB}}
\_def\_puttextB{%
   \_ifvmode
      \_ifdim\_prevdepth>\_zo \_vskip-\_prevdepth \_relax \_fi
      \_nointerlineskip
   \_fi
   \_wd0=\_zo \_ht0=\_zo \_dp0=\_zo
   \_vbox to\_zo{\_kern-\_dimen2 \_hbox to\_zo{\_kern\_dimen1 \_box0\_hss}\_vss}}

\_def\_putpic{\_ea\_ea\_ea\_putpicA\_scantwodimens}
\_def\_putpicA#1#2{\_dimen1=#1sp \_dimen2=#2sp \_ea\_ea\_ea\_putpicB\_scantwodimens}
\_def\_putpicB#1#2#3{{\_setbox0=\_hbox{\_picwidth=#1sp \_picheight=#2sp \_inspic{#3}}\_puttextB}}

\_newbox\_bgbox
\_def\_backgroundpic#1{%
   \_setbox\_bgbox=\_hbox{\_picwidth=\_pdfpagewidth \_picheight=\_pdfpageheight \_inspic{#1}}%
   \_pgbackground={\_copy\_bgbox}
}
\_def\nospec{0pt}
\_public \puttext \putpic \backgroundpic ;
```

`\_circle{`⟨*x*⟩`}{`⟨*y*⟩`}` creates an ellipse with ⟨*x*⟩ axis and ⟨*y*⟩ axis. The origin is in the center.
`\_oval{`⟨*x*⟩`}{`⟨*y*⟩`}{`⟨*roundness*⟩`}` creates an oval with ⟨*x*⟩, ⟨*y*⟩ size and with the given ⟨*roundness*⟩. The real size is bigger by 2⟨*roundness*⟩. The origin is at the left bottom corner.
`\_mv{`⟨*x*⟩`}{`⟨*y*⟩`}{`⟨*curve*⟩`}` moves current point to ⟨*x*⟩, ⟨*y*⟩, creates the ⟨*curve*⟩ and returns the current point back. All these macros are fully expandable and they can be used in the `\pdfliteral` argument.

graphics.opm

```
\def\_circle#1#2{\_expr{.5*(#1)} 0 m
   \_expr{.5*(#1)} \_expr{.276*(#2)} \_expr{.276*(#1)} \_expr{.5*(#2)} 0 \_expr{.5*(#2)} c
   \_expr{-.276*(#1)} \_expr{.5*(#2)} \_expr{-.5*(#1)} \_expr{.276*(#2)} \_expr{-.5*(#1)} 0 c
   \_expr{-.5*(#1)} \_expr{-.276*(#2)} \_expr{-.276*(#1)} \_expr{-.5*(#2)} 0 \_expr{-.5*(#2)} c
   \_expr{.276*(#1)} \_expr{-.5*(#2)} \_expr{.5*(#1)} \_expr{-.276*(#2)} \_expr{.5*(#1)} 0 c h}

\def\_oval#1#2#3{0 \_expr{-(#3)} m \_expr{#1} \_expr{-(#3)} l
    \_expr{(#1)+.552*(#3)} \_expr{-(#3)} \_expr{(#1)+(#3)} \_expr{-.552*(#3)}
                                                           \_expr{(#1)+(#3)} 0 c
    \_expr{(#1)+(#3)} \_expr{#2} l
    \_expr{(#1)+(#3)} \_expr{(#2)+.552*(#3)} \_expr{(#1)+.552*(#3)} \_expr{(#2)+(#3)}
```

```
                                             \_expr{#1} \_expr{(#2)+(#3)} c
    0 \_expr{(#2)+(#3)} l
    \_expr{-.552*(#3)} \_expr{(#2)+(#3)} \_expr{-(#3)} \_expr{(#2)+.552*(#3)}
                                         \_expr{-(#3)} \_expr{#2} c
    \_expr{-(#3)} 0 l
    \_expr{-(#3)} \_expr{-.552*(#3)} \_expr{-.552*(#3)} \_expr{-(#3)}  0 \_expr{-(#3)} c h}

\def\_mv#1#2#3{1 0 0 1 \_expr{#1} \_expr{#2} cm #3 1 0 0 1 \_expr{-(#1)} \_expr{-(#2)} cm}
```

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 boh B. Only boundary stroking is performed after \fcolor=\relax. You cannot combine \fcolor=\relax with \shadow=Y.

graphics.opm

```
\_newdimen \_lwidth
\_def\_fcolor{\_let\_fcolorvalue}
\_def\_lcolor{\_let\_lcolorvalue}
\_def\_shadow{\_let\_shadowvalue}
\_def\_overlapmargins{\_let\_overlapmarginsvalue}
\_def\_ratio{\_isnextchar ={\_ratioA}{\_ratioA=}}
\_def\_ratioA =#1 {\_def\_ratiovalue{#1}}
\_def\_touppervalue#1{\_ifx#1n\_let#1=N\_fi}

\_def\_setflcolors#1{% use only in a group
   \_def\_setcolor##1##2##3{##1 ##2}%
   \_edef#1{\_fcolorvalue}%
   \_def\_setcolor##1##2##3{##1 ##3}%
   \_edef#1{#1\_space\_lcolorvalue\_space}%
}
\_optdef\_inoval[]{\_vbox\_bgroup
   \_roundness=2pt \_fcolor=\Yellow \_lcolor=\Red \_lwidth=.5bp
   \_shadow=N \_overlapmargins=N \_hhkern=0pt \_vvkern=0pt
   \_the\_ovalparams \_relax \_the\_opt \_relax
   \_touppervalue\_overlapmarginsvalue \_touppervalue\_shadowvalue
   \_ifx\_overlapmarginsvalue N%
      \_advance\_hsize by-2\_hhkern \_advance\_hsize by-2\_roundness \_fi
   \_setbox0=\_hbox\_bgroup\_bgroup \_aftergroup\_inovalA \_kern\_hhkern \_let\_next=%
}
\_def\_inovalA{\_egroup % of \setbox0=\hbox\bgroup
   \_ifdim\_vvkern=\_zo \_else \_ht0=\_dimexpr\_ht0+\_vvkern \_relax
                               \_dp0=\_dimexpr\_dp0+\_vvkern \_relax \_fi
   \_ifdim\_hhkern=\_zo \_else \_wd0=\_dimexpr\_wd0+\_hhkern \_relax \_fi
   \_ifx\_overlapmarginsvalue N\_dimen0=\_roundness \_dimen1=\_roundness
   \_else                     \_dimen0=-\_hhkern   \_dimen1=-\_vvkern \_fi
   \_setflcolors\_tmp
   \_hbox{\_kern\_dimen0
      \_vbox to\_zo{\_kern\_dp0
         \_ifx\_shadowvalue N\_else
            \_edef\_tmpb{{\_bp{\_wd0+\_lwidth}}{\_bp{\_ht0+\_dp0+\_lwidth}}{\_bp{\_roundness}}}%
            \_doshadow\_oval
         \_fi
         \_pdfliteral{q \_bp{\_lwidth} w \_tmp
            \_oval{\_bp{\_wd0}}{\_bp{\_ht0+\_dp0}}{\_bp{\_roundness}} \_coc\_space Q}\_vss}%
      \_ht0=\_dimexpr\_ht0+\_dimen1 \_relax \_dp0=\_dimexpr\_dp0+\_dimen1 \_relax
      \_box0
      \_kern\_dimen0}%
   \_egroup % of \vbox\bgroup
}
\_optdef\_incircle[]{\_vbox\_bgroup
   \_ratio=1 \_fcolor=\Yellow \_lcolor=\Red \_lwidth=.5bp
   \_shadow=N \_overlapmargins=N \_hhkern=3pt \_vvkern=3pt
   \_ea\_the \_ea\_circleparams \_space \_relax
   \_ea\_the \_ea\_opt \_space \_relax
   \_touppervalue\_overlapmarginsvalue \_touppervalue\_shadowvalue
   \_setbox0=\_hbox\_bgroup\_bgroup \_aftergroup\_incircleA \_kern\_hhkern \_let\_next=%
}
\_def\_incircleA {\_egroup % of \setbox0=\hbox\bgroup
   \_wd0=\_dimexpr \_wd0+\_hhkern \_relax
   \_ht0=\_dimexpr \_ht0+\_vvkern \_relax \_dp0=\_dimexpr \_dp0+\_vvkern \_relax
   \_ifdim \_ratiovalue\_dimexpr \_ht0+\_dp0 > \_wd0
          \_dimen3=\_dimexpr \_ht0+\_dp0 \_relax  \_dimen2=\_ratiovalue\_dimen3
   \_else \_dimen2=\_wd0 \_dimen3=\_expr{1/\_ratiovalue}\_dimen2 \_fi
   \_setflcolors\_tmp
   \_ifx\_overlapmarginsvalue N\_dimen0=\_zo \_dimen1=\_zo
   \_else \_dimen0=-\_hhkern \_dimen1=-\_vvkern \_fi
   \_hbox{\_kern\_dimen0
      \_ifx\_shadowvalue N\_else
         \_edef\_tmpb{{\_bp{\_dimen2+\_lwidth}}{\_bp{\_dimen3+\_lwidth}}{}}%
         \_doshadow\_circlet
      \_fi
      \_pdfliteral{q \_bp{\_lwidth} w \_tmp \_mv{\_bp{.5\_wd0}}{\_bp{(\_ht0-\_dp0)/2}}
                                      {\_circle{\_bp{\_dimen2}}{\_bp{\_dimen3}} \_coc} Q}%
      \_ifdim\_dimen1=\_zo \_else
           \_ht0=\_dimexpr \_ht0+\_dimen1 \_relax \_dp0=\_dimexpr \_dp0+\_dimen1 \_relax \_fi
      \_box0
      \_kern\_dimen0}
   \_egroup % of \vbox\bgroup
}
\_def\_circlet#1#2#3{\_circle{#1}{#2}}
\_def\_coc{\_ifx\_fcolorvalue\_relax S\_else \_ifdim\_lwidth=0pt f\_else B\_fi\_fi}

\_public \inoval \incircle \ratio \lwidth \fcolor \lcolor \shadow \overlapmargins ;
```

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 2.39.5, 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.

graphics.opm

```
\_def\_addextgstate{\_addpageresource{ExtGState}}

\_public \addextgstate ;
\_def\pageresources{\_pageresources}
\_def\addpageresource{\_addpageresource}
```

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.

graphics.opm

```
\_def\_shadowlevels{9}          % number of layers for blur effect
\_def\_shadowdarknessA{0.025}  % transparency of first shadowlevels/2 layers
\_def\_shadowdarknessB{0.07}   % transparency of second half of layers
\_def\_shadowmoveto{1.8 -2.5}  % vector defines shifting layer (in bp)
\_def\_shadowb{1}              % 2*shadowb = blurring area thickness

\_def\_insertshadowresources{%
   \_addextgstate{op1}{<</ca \_shadowdarknessA>>}%
   \_addextgstate{op2}{<</ca \_shadowdarknessB>>}%
   \_glet\_insertshadowresources=\_relax
}
```

The `\_doshadow{`⟨*curve*⟩`}` does the shadow effect.

graphics.opm

```
\_def\_doshadow#1{\_vbox{%
    \_insertshadowresources
    \_tmpnum=\_numexpr (\_shadowlevels-1)/2 \_relax
    \_edef\_tmpfin{\_the\_tmpnum}%
    \_ifnum\_tmpfin=0 \_def\_shadowb{0}\_def\_shadowstep{0}%
    \_else \_edef\_shadowstep{\_expr{\_shadowb/\_tmpfin}}\_fi
    \_def\_tmpa##1##2##3{\_def\_tmpb
        {#1{##1+2*\_the\_tmpnum*\_shadowstep}{##2+2*\_the\_tmpnum*\_shadowstep}{##3}}}%
    \_ea \_tmpa \_tmpb
    \_def\_shadowlayer{%
        \_ifnum\_tmpnum=0 /op2 gs \_fi
        \_tmpb\_space f
        \_immediateassignment\_advance\_tmpnum by-1
        \_ifnum-\_tmpfin<\_tmpnum
           \_ifx#1\_oval 1 0 0 1 \_shadowstep\_space \_shadowstep\_space cm \_fi
           \_ea \_shadowlayer \_fi
    }%
    \_pdfliteral{q /op1 gs 0 g 1 0 0 1 \_shadowmoveto\_space cm
       \_ifx#1\_circlet 1 0 0 1 \_bp{.5\_wd0} \_bp{(\_ht0-\_dp0)/2} cm
       \_else  1 0 0 1 -\_shadowb\_space -\_shadowb\_space cm \_fi
       \_shadowlayer Q}
}}
```

A generic macro `\_clipinpath`⟨*x*⟩ ⟨*y*⟩ ⟨*curve*⟩ ⟨*text*⟩ declares a clipping path by the ⟨*curve*⟩ shifted by the ⟨*x*⟩, ⟨*y*⟩. The ⟨*text*⟩ is typeset when such clipping path is active. Dimensions are given by bp without the unit here. The macros `\clipinoval` ⟨*x*⟩ ⟨*y*⟩ ⟨*width*⟩ ⟨*height*⟩ `{`⟨*text*⟩`}` and `\clipincircle` ⟨*x*⟩ ⟨*y*⟩ ⟨*width*⟩ ⟨*height*⟩ `{`⟨*text*⟩`}` are defined here. These macros read normal TEX dimensions in their parameters.

graphics.opm

```
\_def\_clipinpath#1#2#3#4{% #1=x-pos[bp], #2=y-pos[bp], #3=curve, #4=text
   \_hbox{\_setbox0=\_hbox{{#4}}%
          \_tmpdim=\_wd0 \_wd0=\_zo
          \_pdfliteral{q \_mv{#1}{#2}{#3 W n}}%
          \_box0\_pdfliteral{Q}\_kern\_tmpdim
   }%
}

\_def\_clipinoval {\_ea\_ea\_ea\_clipinovalA\_scantwodimens}
\_def\_clipinovalA #1#2{%
   \_def\_tmp{{#1/65781.76}{#2/65781.76}}%
   \_ea\_ea\_ea\_clipinovalB\_scantwodimens
}
\_def\_clipinovalB{\_ea\_clipinovalC\_tmp}
\_def\_clipinovalC#1#2#3#4{%
   \_ea\_clipinpath{#1-(#3/131563.52)+(\_bp{\_roundness})}{#2-(#4/131563.52)+(\_bp{\_roundness})}%
   {\_oval{#3/65781.76-(\_bp{2\_roundness})}{#4/65781.76-(\_bp{2\_roundness})}{\_bp{\_roundness}}}%
}
\_def\_clipincircle {\_ea\_ea\_ea\_clipincircleA\_scantwodimens}
\_def\_clipincircleA #1#2{%
   \_def\_tmp{{#1/65781.76}{#2/65781.76}}%
   \_ea\_ea\_ea\_clipincircleB\_scantwodimens
}
\_def\_clipincircleB#1#2{%
   \_ea\_clipinpath\_tmp{\_circle{#1/65781.76}{#2/65781.76}}%
}
\_public \clipinoval \clipincircle ;
```

2.30 The `\table` macro, tables and rules

2.30.1 The boundary declarator `:`

The ⟨*declaration*⟩ part of `\table{`⟨*declaration*⟩`}{`⟨*data*⟩`}` includes column declarators (letters) and other material: the `|` or `(`⟨*cmd*⟩`)`. If the boundary declarator `:` is not used then the boundaries of columns are just before each column declarator with exception of the first one. For example, the declaration `{|c||c(xx)(yy)c}` should be written more exactly using the boundary declarator `:` by `{|c||:c(xx)(yy):c}`. But you can set these boundaries to other places using the boundary declarator `:` explicitly, for example `{|c:||c(xx):(yy)c}`. The boundary declarator `:` can be used only once between each pair of column declarators.

Each table item has its group. The `(`⟨*cmd*⟩`)` are parts of the given table item (depending on the boundary declarator position). If you want to apply a special setting for a given column, you can do this by `(`⟨*setting*⟩`)` followed by column declarator. But if the column is not first, you must use `:(`⟨*setting*⟩`)`. Example. We have three centered columns, the second one have to be in bold font and the third one have to be in red: `\table{c:(\bf)c:(\Red)c}{`⟨*data*⟩`}`

2.30.2 Usage of the `\tabskip` primitive

The value of `\tabskip` primitive is used between all columns of the table. It is glue-type, so it can be stretchable or shrinkable, see next section 2.30.3.

By default, `\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 0 pt.

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 “**p**aragraphs e**x**panded **to**”):

```
\table pxto\hsize {|c|p{\tsize}|}{\crl
 aaa         & Ddkas jd dsjds ds cgha sfgs dd fddzf dfhz xxz
               dras ffg hksd kds d sdjds h sd jd dsjds ds cgha
               sfgs dd fddzf dfhz xxz. \crl
 bb ddd ggg  & Dsjds ds cgha sfgs dd fddzf dfhz xxz
               ddkas jd dsjds ds cgha sfgs dd fddzf. \crl }
```

aaa	Ddkas jd dsjds ds cgha sfgs dd fddzf dfhz xxz dras ffg hksd kds d sdjds h sd jd dsjds ds cgha sfgs dd fddzf dfhz xxz.
bb ddd ggg	Dsjds ds cgha sfgs dd fddzf dfhz xxz ddkas jd dsjds ds cgha sfgs dd fddzf.

The first `c` column is variable width (it gets the width of the most wide item) and the resting space to given `\hsize` is filled by the `p` column.

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

```
\table pxto13cm {r p{3.5\tsize} p{2\tsize} p{\tsize} l}{⟨data⟩}
```

This gives the ratio of widths of individual paragraphs in the table 3.5 : 2 : 1.

2.30.4 `\eqbox`: boxes with equal width across the whole document

The `\eqbox` `[`⟨*label*⟩`]{`⟨*text*⟩`}` behaves like `\hbox{`⟨*text*⟩`}` in the first run of TeX. But the widths of all boxes with the same label are saved to `.ref` file and the maximum box width for each label is calculated at the beginning of the next TeX run. Then `\eqbox` `[`⟨*label*⟩`]{`⟨*text*⟩`}` behaves like `\hbox to` ⟨*dim:label*⟩ `{\hss` ⟨*text*⟩`\hss}`, where ⟨*dim:label*⟩ is the maximum width of all boxes labeled by the same `[`⟨*label*⟩`]`. The documentation of the LaTeX package `eqparbox` includes more information and tips.

The `\eqboxsize` `[`⟨*label*⟩`]{`⟨*dimen*⟩`}` expands to ⟨*dim:label*⟩ if this value is known, else it expands to the given ⟨*dimen*⟩.

The optional parameter `r` or `l` can be written before `[`⟨*label*⟩`]` (for example `\eqbox r[label]{text}`) if you want to put the text to the right or to the left side of the box width.

Try the following example and watch what happens after first TeX run and after the second one.

```
\def\leftitem#1{\par
   \noindent \hangindent=\eqboxsize[items]{2em}\hangafter=1
   \eqbox r[items]{#1 }\ignorespaces}

\leftitem {\bf first}       \lorem[1]
\leftitem {\bf second one} \lorem[2]
\leftitem {\bf final}       \lorem[3]
```

2.30.5 Implementation of the `\table` macro and friends

`table.opm`

```
\_codedecl \table {Macros for tables <2024-11-21>} % preloaded in format
```

The result of the `\table{`⟨*declaration*⟩`}{`⟨*data*⟩`}` macro is inserted into `\_tablebox`. You can change default value if you want by `\let\_tablebox=\vtop` or `\let\_tablebox=\relax`.

`table.opm`

```
\_let\_tablebox=\_vbox
```

We save the `to`⟨*size*⟩ or `pxto`⟨*size*⟩ to `#1` and `\_tableW` sets the `to`⟨*size*⟩ to the `\_tablew` macro. If `pxto`⟨*size*⟩ is used then `\_tablew` is empty and `\_tmpdim` includes given ⟨*size*⟩. The `\_ifpxto` returns true in this case.

The `\table` continues by reading `{`⟨*declaration*⟩`}` in the `\_tableA` macro. Catcodes (for example the `|` character) have to be normal when reading `\table` parameters. This is the reason why we use `\catcodetable` here.

`table.opm`

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

The `\tablinespace` is implemented by enlarging given `\tabstrut` by desired dimension (height and depth too) and by setting `\_lineskip=-2\_tablinespace`. Normal table rows (where no `\hrule` is between them) have normal baseline distance.
The `\_tableA{`⟨*declaration*⟩`}` macro scans the ⟨*declaration*⟩ by `\_scantabdata#1\_relax` and continues by processing `{`⟨*data*⟩`}` by `\_tableB`.

`table.opm`

```
\_def\_tableA#1{\_egroup
   \_the\_thistable \_global\_thistable={}%
   \_ea\_ifx\_ea^\_the\_tabstrut^\_setbox\_tstrutbox=\_null
   \_else \_setbox\_tstrutbox=\_hbox{\_the\_tabstrut}%
          \_setbox\_tstrutbox=\_hbox{\_vrule width\_zo
              height\_dimexpr\_ht\_tstrutbox+\_tablinespace
              depth\_dimexpr\_dp\_tstrutbox+\_tablinespace}%
          \_offinterlineskip
          \_lineskip=-2\_tablinespace
   \_fi
   \_colnum=0 \_let\_addtabitem=\_addtabitemx
   \_def\_tmpa{}\_tabdata={\_colnum1\_relax}\_scantabdata#1\_relax
   \_the\_everytable \_tableB
}
```

The `\_tableB` saves ⟨*data*⟩ to `\_tmpb`. The `#1` parameter is re-scanned in order to set potential `#` characters inside it as others. This is needed because of `\gdef\_tmpb` and next `\replstrings`. They 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=\tabskipl \halign{`⟨*converted declaration*⟩`\tabskip=\tabskipr \cr `⟨*data*⟩`\crcr}`

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

If `pxto` keyword was used, then we set the virtual unit `\tsize` to `-\hsize` first. Then the first attempt of the table is created in box 0. All columns where `p{..\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 `\tsize`s. Then the `\tsize` is re-calculated and the real table is printed by `\halign` in the second pass.

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

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

table.opm

```
\_long\_def\_tableB #1{%
   {{\_catcode`=12 % see OpTeX trick 0117:
     \_directlua{tex.sprint(-1,"\_luaescapestring{\_unexpanded{\_gdef\_tmpb{#1}}}")}}%
    \_tablereplstrings \_edef\_tabskipmid{\_the\_tabskip}\_tabskip=\_tabskipl
    \_ifpxto
      \_edef\_tsizes{\_global\_tsizesum=\_the\_tsizesum \_gdef\_noexpand\_tsizelast{\_tsizelast}}%
      \_tsizesum=\_zo \_def\_tsizelast{0}%
      \_tsize=-\_hsize \_setbox0=\_vbox{\_tablepxpreset \_halign \_tableC}%
      \_advance\_tmpdim by-\_wd0
      \_ifdim \_tmpdim >\_zo \_else \_tsizesum=\_zo \_fi
      \_ifdim \_tsizesum >\_zo \_tsize =\_expr{\_number\_hsize/\_number\_tsizesum}\_tmpdim
      \_else \_tsize=\_zo \_fi
      \_tsizes % restoring values if there is a \table pxto inside a \table pxto.
      \_setbox0=\_null \_halign \_tableC
    \_else
      \_halign\_tablew \_tableC
    \_fi
    \_glet\_tmpb\_undefined
   }\_egroup % \_tablebox\_bgroup is in the \_table macro
}
\_def\_tableC{\_ea{\_the\_tabdata\_tabskip=\_tabskipr\_cr \_scantextokens\_ea{\_tmpb\_crcr}}}
```

`\_tabreplstrings` replaces each `\crl` etc. to `\crcr\crl`. The reason is: we want to use macros that scan its parameter to a delimiter written in the right part of the table item declaration. The `\crcr` cannot be hidden in another macro in this case.

table.opm

```
\_def\_tablereplstrings{%
   \_replstring\_tmpb{\crl}{\_crcr\crl}\_replstring\_tmpb{\crll}{\_crcr\crll}%
   \_replstring\_tmpb{\crli}{\_crcr\crli}\_replstring\_tmpb{\crlli}{\_crcr\crlli}%
   \_replstring\_tmpb{\crlp}{\_crcr\crlp}%
}

\_def\_tablepxpreset{} % can be used to de-activate references to .ref file
\_newbox\_tstrutbox     % strut used in table rows
\_newtoks\_tabdata      % the \halign declaration line
```

The `\_scantabdata` macro converts `\table`'s ⟨*declaration*⟩ to `\halign` ⟨*converted declaration*⟩. The result is stored into `\_tabdata` tokens list. For example, the following result is generated when ⟨*declaration*⟩=`|cr||cl|`.

```
tabdata: \_vrule\_the\_tabiteml{\_hfil#\_unsskip\_hfil}\_the\_tabitemr\_tabstrutA
   &\_the\_tabiteml{\_hfil#\_unsskip}\_the\_tabitemr
                                          \_vrule\_kern\_vvkern\_vrule\_tabstrutA
   &\_the\_tabiteml{\_hfil#\_unsskip\_hfil}\_the\_tabitemr\_tabstrutA
   &\_the\_tabiteml{\_relax#\_unsskip\_hfil}\_the\_tabitemr\_vrule\_tabstrutA
ddlinedata: &\_dditem &\_dditem\_vvitem &\_dditem &\_dditem
```

The second result in the `\_ddlinedata` macro is a template of one row of the table used by `\crli` macro.

table.opm

```
\_def\_scantabdata#1{\_let\_next=\_scantabdata
   \_ifx\_relax#1\_let\_next=\_relax
   \_else\_ifx|#1\_addtabvrule
       \_else\_ifx(#1\_def\_next{\_scantabdataE}%
           \_else\_ifx:#1\_def\_next{\_scantabdataF}%
               \_else\_isinlist{123456789}#1\_iftrue \_def\_next{\_scantabdataC#1}%
                   \_else \_ea\_ifx\_csname _tabdeclare#1\_endcsname \_relax
                       \_ea\_ifx\_csname _paramtabdeclare#1\_endcsname \_relax
                          \_opwarning{tab-declarator "#1" unknown, ignored}%
                       \_else
                          \_def\_next{\_ea\_scantabdataB\_csname _paramtabdeclare#1\_endcsname}\_fi
                       \_else \_def\_next{\_ea\_scantabdataA\_csname _tabdeclare#1\_endcsname}%
   \_fi\_fi\_fi\_fi\_fi\_fi \_next
}
\_def\_scantabdataA#1{\_addtabitem
   \_ea\_addtabdata\_ea{#1\_tabstrutA \_tabskip\_tabskipmid\_relax}\_scantabdata}
\_def\_scantabdataB#1#2{\_addtabitem
   \_ea\_addtabdata\_ea{#1{#2}\_tabstrutA \_tabskip\_tabskipmid\_relax}\_scantabdata}
\_def\_scantabdataC {\_def\_tmpb{}\_afterassignment\_scantabdataD \_tmpnum=}
\_def\_scantabdataD#1{\_loop \_ifnum\_tmpnum>0 \_advance\_tmpnum by-1 \_addto\_tmpb{#1}\_repeat
   \_ea\_scantabdata\_tmpb}
\_def\_scantabdataE#1){\_addtabdata{#1}\_scantabdata}
\_def\_scantabdataF {\_addtabitem\_def\_addtabitem{\_let\_addtabitem=\_addtabitemx}\_scantabdata}
```

The `\_addtabitemx` adds the boundary code (used between columns) to the ⟨*converted declaration*⟩. This code is `\egroup &\bgroup \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/\totcolnum}
\table{ccc}{\showcolnum & \showcolnum & \showcolnum}
```

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.

table.opm

```
\_newcount\_colnum     % number of current column in the table
\_public \colnum ;

\_def\_addtabitemx{\_ifnum\_colnum>0
   \_addtabdata{&}\_addto\_ddlinedata{&\_dditem}\_fi
   \_advance\_colnum by1 \_let\_tmpa=\_relax
   \_ifnum\_colnum>1 \_etoksapp\_tabdata{\_colnum\_the\_colnum\_relax}\_fi}
\_def\_addtabdata{\_toksapp\_tabdata}
```

This code converts `||` or `|` from `\table` ⟨*declaration*⟩ to the ⟨*converted declaration*⟩.

table.opm

```
\_def\_addtabvrule{%
    \_ifx\_tmpa\_vrule \_addtabdata{\_kern\_vvkern}%
       \_ifnum\_colnum=0 \_addto\_vvleft{\_vvitem}\_else\_addto\_ddlinedata{\_vvitem}\_fi
    \_else \_ifnum\_colnum=0 \_addto\_vvleft{\_vvitemA}\_else\_addto\_ddlinedata{\_vvitemA}\_fi\_fi
    \_let\_tmpa=\_vrule \_addtabdata{\_vrule}%
}
\_def\_tabstrutA{\_copy\_tstrutbox}
\_def\_vvleft{}
\_def\_ddlinedata{}
```

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:

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

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

table.opm

```
\_def\_tabdeclarec{\_the\_tabiteml \_hfil{##}\_unsskip \_hfil \_the\_tabitemr}
\_def\_tabdeclarel{\_the\_tabiteml {##}\_unsskip \_hfil\_the\_tabitemr}
\_def\_tabdeclarer{\_the\_tabiteml \_hfil{##}\_unsskip \_the\_tabitemr}
```

The `\_paramtabdeclarep{`⟨*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.

table.opm

```
\_def\_paramtabdeclarep#1{\_hsize=#1\_relax
   \_the\_tabiteml \_tablepar{\_tableparB ##\_tableparC}\_the\_tabitemr
}
\_def\_tablepar{%
   \_ifdim\_hsize<0pt
      \_ifnum\_tsizelast<\_colnum \_global\_advance\_tsizesum by-\_hsize
         \_xdef\_tsizelast{\_the\_colnum}\_fi
      \_let\_tableparbox=\_ignoreit
   \_fi
   \_tableparA \_tableparbox
}
\_let \_tableparbox=\_vtop
\_let \_tableparA=\_empty
\_newdimen \_tsizesum
\_def \_tsizelast{0}
```

The `\_tableparB` initializes the paragraphs inside the table item and `\_tableparC` closes them. They are used in the `\_paramtabdeclarep` macro. The first paragraph is no indented.

table.opm

```
\_def\_tableparB{%
   \_baselineskip=\_normalbaselineskip \_lineskiplimit=\_zo \_noindent
   \_unless\_ifx\_tabstrutA\_empty \_raise\_ht\_tstrutbox\_null \_fi
   \_hskip\_zo \_relax
}
\_def\_tableparC{%
   \_unsskip
   \_unless\_ifx\_tabstrutA\_empty
      \_ifvmode\_vskip\_dp\_tstrutbox \_else\_lower\_dp\_tstrutbox\_null\_fi
   \_fi
}
```

Users put optional spaces around the table item typically, i.e. they write `& text &` instead `&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&`.

table.opm

```
\_def\_unsskip{\_ifmmode\_else\_ifdim\_lastskip>\_zo \_unskip\_fi\_fi}
```

The `\fL`, `\fR`, `\fC` and `\fX` macros only do special parameters settings for paragraph building algorithm.

table.opm

```
\_let\_fL=\_raggedright
\_def\_fR{\_leftskip=0pt plus 1fill \_relax}
\_def\_fC{\_leftskip=0pt plus1fill \_rightskip=0pt plus 1fill \_relax}
\_def\_fX{\_leftskip=0pt plus1fil  \_rightskip=0pt plus-1fil \_parfillskip=0pt plus2fil \_relax}
\_public \fL \fR \fC \fX ;
```

The `\fS` macro is more tricky. The `\_tableparbox` isn't printed immediately, but `\setbox2=` is prefixed by the macro `\_tableparA`, which is empty by default (used in `\_tablepar`). The `\_tableparD` is processed after the box is set: it checks if there is only one line and prints `\hbox to\hsize{\hfil`⟨*this line*⟩`\hfil}` in this case. In other cases, the box2 is printed.

table.opm

```
\_def\_fS{\_relax
   \_ifdim\_hsize<0pt \_else \_def\_tableparA{\_setbox2=}\_fi
   \_addto\_tableparC{\_aftergroup\_tableparD}%
}
\_def\_tableparD{\_setbox0=\_vbox{\_unvcopy2 \_unskip \_global\_setbox1=\_lastbox}%
   \_ifdim\_ht0>0pt \_box2 \_setbox0=\_box1
   \_else \_hbox to\_hsize{\_hfil \_unhbox1\_unskip\_unskip\_hfil}\_setbox0=\_box2 \_fi
}
\_public \fS ;
```

The family of `\_cr*` macros `\crl`, `\crll`, `\crli`, `\crlli`, `\crlp` and `\tskip` ⟨*dimen*⟩ is implemented here. The `\_zerotabrule` is used to suppress the negative `\lineskip` declared by `\tablinespace`.

table.opm

```
\_def\_crl{\_crcr\_noalign{\_hrule}}
\_def\_crll{\_crcr\_noalign{\_hrule\_kern\_hhkern\_hrule}}
\_def\_zerotabrule {\_noalign{\_hrule height\_zo width\_zo depth\_zo}}

\_def\_crli{\_crcr \_zerotabrule \_omit
   \_gdef\_dditem{\_omit\_tablinefil}\_gdef\_vvitem{\_kern\_vvkern\_vrule}\_gdef\_vvitemA{\_vrule}%
   \_vvleft\_tablinefil\_ddlinedata\_crcr \_zerotabrule}
\_def\_crlli{\_crli\_noalign{\_kern\_hhkern}\_crli}
\_def\_tablinefil{\_leaders\_hrule\_hfil}

\_def\_crlp#1{\_crcr \_zerotabrule \_noalign{\_kern-\_drulewidth}%
   \_omit \_xdef\_crlplist{#1}\_xdef\_crlplist{,\_ea}\_ea\_crlpA\_crlplist,\_fin,%
   \_global\_tmpnum=0 \_gdef\_dditem{\_omit\_crlpD}%
   \_gdef\_vvitem{\_kern\_vvkern\_kern\_drulewidth}\_gdef\_vvitemA{\_kern\_drulewidth}%
   \_vvleft\_crlpD\_ddlinedata \_global\_tmpnum=0 \_crcr \_zerotabrule}
\_def\_crlpA#1,{\_ifx\_fin#1\_else \_crlpB#1-\_fin,\_ea\_crlpA\_fi}
\_def\_crlpB#1#2-#3,{\_ifx\_fin#3\_xdef\_crlplist{\_crlplist#1#2,}\_else\_crlpC#1#2-#3,\_fi}
\_def\_crlpC#1-#2-#3,{\_tmpnum=#1\_relax
   \_loop \_xdef\_crlplist{\_crlplist\_the\_tmpnum,}\_ifnum\_tmpnum<#2\_advance\_tmpnum by1 \_repeat}
\_def\_crlpD{\_incr\_tmpnum \_edef\_tmpa{\_noexpand\_isinlist\_noexpand\_crlplist{,\_the\_tmpnum,}}%
   \_tmpa\_iftrue \_kern-\_drulewidth \_tablinefil \_kern-\_drulewidth\_else\_hfil \_fi}

\_def\_tskip{\_afterassignment\_tskipA \_tmpdim}
\_def\_tskipA{\_gdef\_dditem{}\_gdef\_vvitem{}\_gdef\_vvitemA{}\_gdef\_tabstrutA{}%
    \_vbox to\_tmpdim{}\_ddlinedata \_crcr
    \_zerotabrule \_noalign{\_gdef\_tabstrutA{\_copy\_tstrutbox}}}

\_public \crl \crll \crli \crlli \crlp \tskip ;
```

The `\mspan{`⟨*number*⟩`}[`⟨*declaration*⟩`]{`⟨*text*⟩`}` macro generates similar `\omit\span\omit\span` sequence as plain TeX macro `\multispan`. Moreover, it uses `\_scantabdata` to convert ⟨*declaration*⟩ from `\table` syntax to `\halign` syntax.

table.opm

```
\_def\_mspan{\_omit \_afterassignment\_mspanA \_mscount=}
\_def\_mspanA[#1]#2{\_loop \_ifnum\_mscount>1 \_cs{_span}\_omit \_advance\_mscount-1 \_repeat
   \_count1=\_colnum \_colnum=0 \_def\_tmpa{}\_tabdata={}\_scantabdata#1\_relax
   \_colnum=\_count1 \_setbox0=\_vbox{\_halign\_ea{\_the\_tabdata\_cr#2\_cr}%
   \_global\_setbox8=\_lastbox}%
   \_setbox0=\_hbox{\_unhbox8 \_unskip \_global\_setbox8=\_lastbox}%
   \_unhbox8 \_ignorespaces}
\_public \mspan ;
```

The `\vspan`⟨*number*⟩`{`⟨*text*⟩`}` implementation is here. We need to lower the box by

`(`⟨*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.

table.opm

```
\_def\_vspan#1#2#{\_vspanA{#1#2}}
\_def\_vspanA#1#2{\_vtop to\_zo{\_hbox{\_lower \_dimexpr
   #1\_dimexpr(\_ht\_tstrutbox+\_dp\_tstrutbox)/2\_relax
    -\_dimexpr(\_ht\_tstrutbox+\_dp\_tstrutbox)/2\_relax \_hbox{#2}}\_vss}}
\_public \vspan ;
```

The parameters of primitive `\vrule` and `\hrule` keeps the rule “last wins”. If we re-define `\hrule` to `\_orihrule height1pt` then each usage of redefined `\hrule` uses `1pt` height if this parameter isn’t overwritten by another following `height` parameter. This principle is used for settings another default rule thickness than 0.4 pt by the macro `\rulewidth`.

table.opm
```
\_newdimen\_drulewidth  \_drulewidth=0.4pt
\_let\_orihrule=\_hrule  \_let\_orivrule=\_vrule
\_def\_rulewidth{\_afterassignment\_rulewidthA \_drulewidth}
\_def\_rulewidthA{\_edef\_hrule{\_orihrule height\_drulewidth}%
                  \_edef\_vrule{\_orivrule width\_drulewidth}%
                  \_let\_rulewidth=\_drulewidth
                  \_public \vrule \hrule \rulewidth;}
\_public \rulewidth ;
```

The `\frame{`⟨*text*⟩`}` uses “`\vbox` in `\vtop`” trick in order to keep the baseline of the internal text at the same level as outer baseline. User can write `\frame{abcxyz}` in normal paragraph line, for example and gets the expected result: $\boxed{\text{abcxyz}}$. The internal margins are set by `\vvkern` and `\hhkern` parameters.

table.opm
```
\_long\_def\_frame#1{%
   \_hbox{\_vrule\_vtop{\_vbox{\_hrule\_kern\_vvkern
      \_hbox{\_kern\_hhkern{#1}\_kern\_hhkern}%
   }\_kern\_vvkern\_hrule}\_vrule}}
\_public \frame ;
```

`\eqbox` and `\eqboxsize` are implemented here. The widths of all `\eqbox`es are saved to the `.ref` file in the format `\_Xeqbox{`⟨*label*⟩`}{`⟨*size*⟩`}`. The `.ref` file is read again and maximum box width for each ⟨*label*⟩ is saved to `\_eqb:`⟨*label*⟩.

table.opm
```
\_def\_Xeqbox#1#2{%
   \_ifcsname _eqb:#1\_endcsname
      \_ifdim #2>\_cs{_eqb:#1}\_relax \_sdef{_eqb:#1}{#2}\_fi
   \_else \_sdef{_eqb:#1}{#2}\_fi
}
\_def\_eqbox #1[#2]#3{\_setbox0=\_hbox{{#3}}%
   \_openref \_immediate\_wref \_Xeqbox{{#2}{\_the\_wd0}}%
   \_ifcsname _eqb:#2\_endcsname
       \_hbox to\_cs{_eqb:#2}{\_ifx r#1\_hfill\_fi\_hss\_unhbox0\_hss\_ifx l#1\_hfill\_fi}%
   \_else \_box0 \_fi
}
\_def\_eqboxsize [#1]#2{\_trycs{_eqb:#1}{#2}}

\public \eqbox \eqboxsize ;
```

2.31 Balanced multi-columns

multicolumns.opm
```
\_codedecl \begmulti {Balanced columns <2024-10-31>} % preloaded in format
```

`\_betweencolumns` or `\_leftofcolumns` or `\_rightofcolumns` include a material printed between columns or left of all columns or right of all columns respectively. The `\_betweencolumns` must include a stretchability or a material with exactly `\colsep` width. You can redefine these macros. For example the rule between columns can be reached by `\_def\_betweencolumns{\hss\vrule\hss}`.
`\_multiskip` puts its material at the start and at the end of `\begmulti`...`\endmulti`.

multicolumns.opm
```
\_def\_betweencolumns{\_hss} \_def\_leftofcolumns{} \_def\_rightofcolumns{}
\_def\_multiskip{\_medskip}   % space above and below \begmulti...\endmulti
```

The code used here is documented in detail in the “TEXbook naruby”, pages 244–246, free available, http://petr.olsak.net/tbn.html, but in Czech. Roughly speaking, macros complete all material between `\begmulti`⟨*num-columns*⟩ and `\endmulti` into one `\vbox 6`. Then the macro measures the amount of free space at the current page using `\pagegoal` and `\pagtotal` and does `\vsplit` of `\vbox 6` to columns with a height of such free space. This is done only if we have enough amount of material in `\vbox 6` to fill the full page by columns. This is repeated in a loop until we have less amount of material in `\vbox 6`. Then we run `\_balancecolumns` which balances the last part of the columns. Each part of printed material is distributed to the main vertical list as `\hbox{`⟨*columns*⟩`}` and we need not do any change in the output routine.

If you have paragraphs in `\begmulti`... `\endmulti` environment then you may say `\raggedright` inside this environment and you can re-assign `\widowpenalty` and `\clubppenalty` (they are set to 10000 in OpTEX).

multicolumns.opm

```
\_def\_begmulti #1 {\_par\_bgroup\_wipeepar
   \_ifnum\_lastpenalty>10000 \_vskip4.5\_baselineskip\_penalty9999 \_vskip-4.5\_baselineskip \_fi
   \_multiskip \_def\_Ncols{#1}
   \_setbox6=\_vbox\_bgroup\_bgroup \_let\_setxhsize=\_relax \_penalty-99
   %% \hsize := column width = (\hsize+\colsep) / n - \colsep
   \_setbox0=\_hbox{\_leftofcolumns\_rightofcolumns}%
   \_advance\_hsize by-\_wd0 \_advance\_hsize by\_colsep
   \_divide\_hsize by\_Ncols  \_advance\_hsize by-\_colsep
}
\_def\_endmulti{\_vskip-\_prevdepth\_vfil
   \_ea\_egroup\_ea\_egroup\_ea\_baselineskip\_the\_baselineskip\_relax
   \_dimen0=.8\_maxdimen \_tmpnum=\_dimen0 \_divide\_tmpnum by\_baselineskip
   \_splittopskip=\_baselineskip
   \_setbox1=\_vsplit6 to0pt % initialize first \splittopskip in \box6
   %% \dimen1 := the free space on the page
   \_penalty0 % initialize \_pageoal
   \_ifdim\_pagegoal=\_maxdimen \_setcolsize\_vsize
   \_else \_setcolsize{\_dimexpr\_pagegoal-\_pagetotal}\_fi
   \_ifdim \_dimen1<2\_baselineskip \_vfil\_break \_setcolsize\_vsize \_fi
   \_setsplitsize % sets \_dimen0 = \_ht6
   \_divide\_dimen0 by\_Ncols \_relax
   %% split the material to more pages?
   \_ifdim \_dimen0>\_dimen1 \_splitpart
   \_else \_balancecolumns \_fi  % only balancing
   \_multiskip \_egroup
}
```

Splitting columns...

multicolumns.opm

```
\_def\_makecolumns{\_bgroup % full page, destination height: \dimen1
   \_vbadness=20000 \_splitmaxdepth=\_maxdepth \_dimen6=\_wd6
   \_createcolumns
   \_printcolumns
   \_egroup
}
\_def\_splitpart{%
   \_makecolumns % full page
   \_vskip 0pt plus 1fil minus\_baselineskip \_break
   \_setsplitsize % sets \_dimen0 = \_ht6
   \_divide\_dimen0 by\_Ncols \_relax
   \_ifx\_balancecolumns\_flushcolumns \_advance\_dimen0 by-.5\_vsize \_fi
   \_setcolsize\_vsize \_dimen2=\_dimen1
   \_advance\_dimen2 by-\_baselineskip
   %% split the material to more pages?
   \_ifvoid6 \_else
      \_ifdim \_dimen0>\_dimen2 \_ea\_ea\_ea \_splitpart
      \_else \_balancecolumns % last balancing
   \_fi \_fi
}
```

Final balancing of the columns.

multicolumns.opm

```
\_def\_balancecolumns{\_bgroup \_setbox7=\_copy6 % destination height: \dimen0
   \_ifdim\_dimen0>\_baselineskip \_else \_dimen0=\_baselineskip \_fi
   \_vbadness=20000 \_dimen6=\_wd6 \_dimen1=\_dimen0
   \_def\_trybalance{\_createcolumns
      \_ifvoid6 \_else
         \_advance \_dimen1 by.2\_baselineskip
         \_setbox6=\_copy7
         \_ea \_trybalance \_fi}\_trybalance
   \_printcolumns
   \_egroup
}
```

`\_setcolsize`⟨*dimen*⟩ sets initial value `\dimen1=`⟨*size*⟩ which is used as height of columns at given page. The correction `\splittopskip`−`\topskip` is done if the columns start at the top of the page.

`\_createcolumns` prepares columns with given height `\dimen1` side by side to the `\box1`.
`\_printcolumns` prints the columns prepared in `\box1`. The first `\hbox{}` moves typesetting point to the next baseline. Next negative skip ensures that the first line from split columns is at this position.

multicolumns.opm

```
\_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 ;
```

The accumulated `\box6` (before its splitting to pages and columns) can be very large and we cannot measure it with classical TeX methods because we can get “dimension too lagre” error or arithmetic overflow. The TeX limit for dimensions is about 5.75 m. For example, the `\box6` of the index in this document has about 6.5 m height.

The `\_rawht`⟨*box-num*⟩ macro solves this problem. It expands to the height plus depth of given `\vbox` as an integer in points (i.e. the the fraction part of points is stripped and unit isn’t appended). The limit 2^{31} of such an integer allows us to measure boxes with many kilometers height.

The trick used in the `\_setsplitsize` macro sets `\dimen0` to the maximum of `.8\maxdimen` and actual `\ht6`, because 13100 is `.8\maxdimen` in points. This is sufficient because if the `\dimen0` is big then we get n columns for full page. This is not the last page where exact calculations for balancing is needed.

multicolumns.opm

```
\_def\_setsplitsize{%
   \_ifnum \_rawht6 > 13100 \_dimen0 = .8\_maxdimen
   \_else \_dimen0=\_ht6 \_def\_rawht6{0}\_fi
}
\_def\_rawht{\_directlua{optex.raw_ht()}}
```

2.32 Citations, bibliography

2.32.1 Macros for citations and bibliography preloaded in the format

cite-bib.opm

```
\_codedecl \cite {Cite, Bibliography <2021-04-13>} % preloaded in format
```

Registers used by `\cite`, `\bib` macros are declared here. The `\bibnum` counts the bibliography items from one. The `\bibmark` is used when `\nonumcitations` is set.

cite-bib.opm

```
\_newcount\_bibnum                    % the bibitem counter
\_newtoks\_bibmark                    % the bibmark used if \nonumcitations
\_newcount\_lastcitenum \_lastcitenum=0  % for \shortcitations
\_public \bibnum \bibmark ;
```

`\_bibp` expands to `\bibpart/`. By default, `\bibpart` is empty, so internal links are in the form `cite:/`⟨*number*⟩. If `\bibpart` is set to ⟨*bibpart*⟩, then internal links are `cite:`⟨*bibpart*⟩`/`⟨*number*⟩.

cite-bib.opm

```
\_def\_bibp{\_the\_bibpart/}           % unique name for each bibliography list
```

`\cite` `[`⟨*label*⟩`,`⟨*label*⟩`,...,`⟨*label*⟩`]` manages ⟨*labes*⟩ using `\_citeA` and prints `[`⟨*bib-marks*⟩`]` using `\_printsavedcites`.
`\nocite` `[`⟨*label*⟩`,`⟨*label*⟩`,...,`⟨*label*⟩`]` only manages ⟨*labels*⟩ but prints nothing.
`\rcite` `[`⟨*label*⟩`,`⟨*label*⟩`,...,`⟨*label*⟩`]` behaves like `\cite` but prints ⟨*bib-marks*⟩ without brackets.
`\ecite` `[`⟨*label*⟩`]{`⟨*text*⟩`}` behaves like `\rcite [`⟨*label*⟩`]` but prints ⟨*text*⟩ instead ⟨*bib-mark*⟩. The ⟨*text*⟩

is hyperlinked like ⟨*bib-marks*⟩ when `\cite` or `\rcite` is used. The empty internal macro `\_savedcites` will include the ⟨*bib-marks*⟩ list to be printed. This list is set by `\_citeA` inside a group and it is used by `\_printsavedcites` in the same group. Each `\cite`/`\rcite`/`\ecite` macro starts from empty list of ⟨*bib-marks*⟩ because new group is opened.

cite-bib.opm

```
\_def\_cite[#1]{{\_citeA#1,,,[\_printsavedcites]}}
\_def\_nocite[#1]{{\_citeA#1,,,}}
\_def\_rcite[#1]{{\_citeA#1,,,\_printsavedcites}}
\_def\_ecite[#1]{\_bgroup\_citeA#1,,,\_ea\_eciteB\_savedcites;}
\_def\_eciteB#1,#2;#3{\_if?#1\_relax #3\_else \_ilink[cite:\_bibp#1]{#3}\_fi\_egroup}
\_def\_savedcites{}

\_public \cite \nocite \rcite \ecite ;
```

⟨*bib-marks*⟩ may be numbers or a special text related to cited bib-entry. It depends on `\nonumcitations` and on used bib-style. The mapping from ⟨*label*⟩ to ⟨*bib-mark*⟩ is done when `\bib` or `\usebib` is processed. These macros store the information to `\_Xbib{`⟨*bibpart*⟩`}{`⟨*label*⟩`}{`⟨*number*⟩`}{`⟨*nonumber*⟩`}` where ⟨*number*⟩ and ⟨*nonumber*⟩ are two variants of ⟨*bib-mark*⟩ (numbered or text-like). This information is read from `.ref` file and it is saved to macros `\_bib:`⟨*bibpart*⟩`/`⟨*label*⟩ and `\_bim:`⟨*bibpart*⟩`/`⟨*number*⟩. First one includes ⟨*number*⟩ and second one includes ⟨*nonumber*⟩. The `\_lastbn:`⟨*bibpart*⟩ macro includes last number of bib-entry used in the document with given ⟨*bibpart*⟩. A designer can use it to set appropriate indentation when printing the list of all bib-entries.

cite-bib.opm

```
\_def\_Xbib#1#2#3#4{\_sxdef{_bib:#1/#2}{\_bibnn{#3}&}%
   \_if^#4^\_else\_sxdef{_bim:#1/#3}{#4}\_fi\_sxdef{_lastbn:#1}{#3}}
```

`\_citeA` ⟨*label*⟩`,` processes one label from the list of labels given in the parameter of `\cite`, `\nocite`, `\rcite` or `\ecite` macros. It adds the ⟨*label*⟩ to a global list `\_ctlst:`⟨*bibpart*⟩`/` which will be used by `\usebib` (it must know what ⟨*labels*⟩ are used in the document to pick-up only relevant bib-entries from the database. Because we want to save space and to avoid duplications of ⟨*label*⟩ in the `\_ctlst:`⟨*bibpart*⟩`/`, we distinguish four cases:

- ⟨*label*⟩ was not declared by `\_Xbib` before and it is first such a ⟨*label*⟩ in the document: Then `\_bib:`⟨*bibpart*⟩`/`⟨*label*⟩ is undefined and we save label using `\_addcitelist`, write warning on the terminal and define `\_bib:`⟨*bibpart*⟩`/`⟨*label*⟩ as empty.
- ⟨*label*⟩ was not declared by `\_Xbib` before but it was used previously in the document: Then `\_bib:`⟨*bibpart*⟩`/`⟨*label*⟩ is empty and we do nothing (only data to `\_savedcites` are saved).
- ⟨*label*⟩ was declared by `\_Xbib` before and it is first such ⟨*label*⟩ used in the document: Then `\_bib:`⟨*bibpart*⟩`/`⟨*label*⟩ includes `\_bibnn{`⟨*number*⟩`}&` and we test this case by the command `\if &\_bibnn{`⟨*number*⟩`}&`. This is true when `\_bibnn{`⟨*number*⟩`}` expands to empty. The ⟨*label*⟩ is saved by `\_addcitelist` and `\_bib:`⟨*bibpart*⟩`/`⟨*label*⟩ is re-defined directly as ⟨*number*⟩.
- ⟨*label*⟩ was declared by `\_Xbib` and it was used previously in the document. Then we do nothing (only data to `\_savedcites` are saved.

The `\_citeA` macro runs repeatedly over the whole list of ⟨*labels*⟩.

cite-bib.opm

```
\_def\_citeA #1#2,{\_if#1,\_else
   \_if *#1\_addcitelist{*}\_sxdef{_bib:\_bibp*}{}\_ea\_skiptorelax \_fi
   \_ifcsname _bib:\_bibp#1#2\_endcsname \_else
      \_addcitelist{#1#2}%
      \_opwarning{{\_the\_bibpart} \_noexpand\cite [#1#2] unknown. Try to TeX me again}\_openref
      \_incr\_unresolvedrefs
      \_addto\_savedcites{?,}\_def\_sortcitesA{}\_lastcitenum=0
      \_ea\_gdef \_csname _bib:\_bibp#1#2\_endcsname {}%
      \_ea\_skiptorelax \_fi
   \_ea\_ifx \_csname _bib:\_bibp#1#2\_endcsname \_empty
      \_addto\_savedcites{?,}\_def\_sortcitesA{}\_lastcitenum=0
      \_ea\_skiptorelax \_fi
   \_def\_bibnn##1{}%
   \_if &\_csname _bib:\_bibp#1#2\_endcsname
      \_def\_bibnn##1##2{##1}%
      \_addcitelist{#1#2}%
      \_sxdef{_bib:\_bibp#1#2}{\_csname _bib:\_bibp#1#2\_endcsname}%
   \_fi
   \_edef\_savedcites{\_savedcites \_csname _bib:\_bibp#1#2\_endcsname,}%
   \_relax
   \_ea\_citeA\_fi
}
\_let\_bibnn=\_relax
```

Because we implement possibility of more independent bibliography lists distinguished by ⟨*bibpart*⟩, the _addcitelist{⟨*label*⟩} macro must add the ⟨*label*⟩ to given _ctlst:⟨*bibpart*⟩/.
When _addcitelist is processed before \usebib, then _citeI[⟨*label*⟩] is added. \usebib will use this list for selecting right records from .bib file. Then \usebib sets _ctlst:⟨*bibpart*⟩/ to _write.
If _addcitelist is processed after \usebib, then _Xcite{⟨*bibpart*⟩/}{⟨*label*⟩} is saved to the .ref file. The _Xcite creates _ctlstB:⟨*bibpart*⟩/ as a list of saved _citeI[⟨*label*⟩]. Finally, \usebib concats both lists _ctlst:⟨*bibpart*⟩/ and _ctlstB:⟨*bibpart*⟩/ in the second TeX run.

cite-bib.opm

```
\_def\_addcitelist#1{%
   \_unless \_ifcsname _ctlst:\_bibp\_endcsname \_sxdef{_ctlst:\_bibp}{}\_fi
   \_ea \_ifx \_csname _ctlst:\_bibp\_endcsname \_write
       \_openref \_immediate\_wref\_Xcite{{\_bibp}{#1}}%
   \_else \_global \_ea\_addto \_csname _ctlst:\_bibp\_endcsname {\_citeI[#1]}\_fi
}
\_def\_Xcite#1#2{%
   \_unless \_ifcsname _ctlstB:#1\_endcsname \_sxdef{_ctlstB:#1}{}\_fi
   \_global \_ea\_addto \_csname _ctlstB:#1\_endcsname {\_citeI[#2]}%
}
```

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).

cite-bib.opm

```
\_def\_printsavedcites{\_sortcitesA
   \_chardef\_tmpb=0 \_ea\_citeB\_savedcites,%
   \_ifnum\_tmpb>0 \_printdashcite{\_the\_tmpb}\_fi
}
\_def\_sortcitesA{}
\_def\_sortcitations{%
  \_def\_sortcitesA{\_edef\_savedcites{300000,\_ea}\_ea\_sortcitesB\_savedcites,%
                    \_def\_tmpa####1300000,{\_def\_savedcites{####1}}\_ea\_tmpa\_savedcites}%
}
\_def\_sortcitesB #1,{\_if $#1$%
  \_else
     \_mathchardef\_tmpa=#1
     \_edef\_savedcites{\_ea}\_ea\_sortcitesC \_savedcites\_end
     \_ea\_sortcitesB
  \_fi
}
\_def\_sortcitesC#1,{\_ifnum\_tmpa<#1\_edef\_tmpa{\_the\_tmpa,#1}\_ea\_sortcitesD
                   \_else\_edef\_savedcites{\_savedcites#1,}\_ea\_sortcitesC\_fi}
\_def\_sortcitesD#1\_end{\_edef\_savedcites{\_savedcites\_tmpa,#1}}

\_def\_citeB#1,{\_if$#1$\_else
   \_if?#1\_relax??%
      \_else
      \_ifnum\_lastcitenum=0   % only comma separated list
         \_printcite{#1}%
      \_else
         \_ifx\_citesep\_empty  % first cite item
            \_lastcitenum=#1\_relax
            \_printcite{#1}%
         \_else              % next cite item
            \_advance\_lastcitenum by1
            \_ifnum\_lastcitenum=#1\_relax % cosecutive cite item
               \_mathchardef\_tmpb=\_lastcitenum
            \_else  % there is a gap between cite items
               \_lastcitenum=#1\_relax
               \_ifnum\_tmpb=0 % previous items were printed
                  \_printcite{#1}%
               \_else
                  \_printdashcite{\_the\_tmpb}\_printcite{#1}\_chardef\_tmpb=0
   \_fi\_fi\_fi\_fi\_fi
   \_ea\_citeB\_fi
}
\_def\_shortcitations{\_lastcitenum=1 }

\_def\_printcite#1{\_citesep
   \_ilink[cite:\_bibp#1]{\_citelinkA{#1}}\_def\_citesep{,\_hskip.2em\_relax}}
\_def\_printdashcite#1{\_ifmmode-\_else\_hbox{--}\_fi\_ilink[cite:\_bibp#1]{\_citelinkA{#1}}}
\_def\_citesep{}

\_def\_nonumcitations{\_lastcitenum=0\_def\_sortcitesA{}\_def\_etalchar##1{$^{##1}$}%
   \_def\_citelinkA##1{\_trycs{_bim:\_bibp##1}
       {##1\_opwarning{\_noexpand\nonumcitations + empty bibmark. Maybe bad bib-style}}}%
}
\_def\_citelinkA{}

\_public \nonumcitations \sortcitations \shortcitations ;
```

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`s from one by `\bibnum` counter and saves `\_Xbib{`⟨*bibpart*⟩`}{`⟨*label*⟩`}{`⟨*number*⟩`}{`⟨*nonumber*⟩`}` into `.ref` file immediately using `\_wbib{`⟨*label*⟩`}{`⟨*number*⟩`}{`⟨*nonumber*⟩`}`. This is the core of creation of mapping from ⟨*labels*⟩ to ⟨*number*⟩ and ⟨*nonumber*⟩.
`\_bibA` and `\_bibB` implement the scanner of the optional argument with the `\bibmark`.
`\_bibgl` is `\relax` by default but `\slides` do `\let\_bibgl=\_global`.
`\_dbib{`⟨*label*⟩`}` creates destination for hyperlinks.

cite-bib.opm

```
\_def\_bib[#1]{\_def\_tmp{\_isnextchar={\_bibA[#1]}{\_bibmark={}\_bibB[#1]}}%
   \_nospaceafter\_tmp} % ignore optional space
\_def\_bibA[#1]=#2{\_bibmark={#2}\_bibB[#1]}
\_def\_bibB[#1]{\_par \_bibskip
   \_bibgl\_advance\_bibnum by1
   \_noindent \_def\_tmpb{#1}\_dbib{#1}\_wbib{#1}{\_the\_bibnum}{\_the\_bibmark}%
   \_printbib \_ignorespaces
}
\_def\_dbib#1{\_dest[cite:\_bibp\_the\_bibnum]\_printlabel{#1}}
\_def\_wbib#1#2#3{%
   \_ifx\_wref\_wrefrelax\_else \_immediate\_wref\_Xbib{{\_the\_bibpart}{#1}{#2}{#3}}\_fi
   \_unless \_ifcsname bib:\_bibp#1\_endcsname \_Xbib{\_the\_bibpart}{#1}{#2}{#3}\_fi
}
\_let\_bibgl=\_relax

\_public \bib ;
```

The `\_printbib` prints the bib-entry itself. You can re-define it if you want a different design. The `\_pritbib` starts in horizontal mode after `\noindent` and after the eventual hyperlink destination is inserted. By default, the `\_printbib` sets the indentation by `\hangindent` and prints numeric ⟨*bib-marks*⟩ by `\llap{[\the\bibnum]}` If `\nonumcitations` then the `\_citelinkA` is not empty and ⟨*bib-marks*⟩ (`\the\bibnum` nor `\the\bibmark`) are not printed. The text of bib-entry follows. User can create this text manually using `\bib` command or it is generated automatically from a `.bib` database by `\usebib` command.

The vertical space between bib-entries is controlled by `\_bibskip` macro.

cite-bib.opm

```
\_def \_printbib {\_hangindent=\_iindent
   \_ifx\_citelinkA\_empty \_hskip\_iindent \_llap{[\_the\_bibnum] }\_fi
}
\_def \_bibskip {\_ifnum\_bibnum>0 \_smallskip \_fi}
```

The `\usebib` command is implemented in `usebib.opm` file which is loaded when the `\usebib` command is used first. The `usebib.opm` file loads the `librarian.tex` for scanning the `.bib` files. See the section 2.32.2, where the file `usebib.opm` is documented.

cite-bib.opm

```
\_def\_usebib{\_par \_opinput {usebib.opm} \_usebib}
\_def\usebib{\_usebib}
```

`\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.

cite-bib.opm

```
\_def\_nobibwarnlist{,}
\_def\_nobibwarning[#1]{\_global\_addto\_nobibwarnlist{#1,}}
\_public \nobibwarning ;
```

2.32.2 The `\usebib` command

The file `usebib.opm` implements the command `\usebib/`⟨*sorttype*⟩ `(`⟨*style*⟩`)` ⟨*bibfiles*⟩ where ⟨*sorttype*⟩ is one letter `c` (references ordered by citation order in the text) or `s` (references sorted usually by authors and years), ⟨*style*⟩ is the part of the name `bib-`⟨*style*⟩`.opm` of the style file and ⟨*bibfiles*⟩ are one or more `.bib` file names without suffix separated by comma without space. Example:

```
\usebib/s (simple) mybase,yourbase
```

This command reads the ⟨*bibfiles*⟩ directly and creates the list of bibliographic references (only those declared by `\cite[]` or `\nocite[]` in the document). The formatting of such references is defined in the style file.

The principle “first entry wins” is used. Suppose `\usebib/s (simple) local,global`. If an entry with the same label is declared in `local.bib` and in `global.bib` too then the first wins. So, you can set exceptions in your `local.bib` file for your document.

The `bib-`⟨*style*⟩`.opm` file declares entry types (like `@BOOK`, `@ARTICLE`) and declares their mandatory and optional fields (like `author`, `title`). When a mandatory field is missing in an entry in the `.bib` file then a warning is printed on the terminal about it. You can suppress such warnings by command `\nobibwarning [`⟨*bib-labels*⟩`]`, where ⟨*bib-labels*⟩ is a comma-separated list of labels (without spaces) where missing mandatory fields will be no warned.

Users may redefine declarations and the formatting rules given by the macros from the style file. Such a re-definition have to be included in the `\bibtexhook` token list, because the `\usebib` macro opens group, reads the macros from the style file, then executes `\bibtexhook` (it is empty by default), then reads data from the `.bib` files, then prints the desired records and finally, it closes the group.

For example, `\bibtexhook={\oldaccents}` can be set if your old `.bib` files use an obscure notation for accents like `{\"o}`. Recommendation: converting such old `.bib` files to Unicode encoding is much more conceptual solution of this problem.

2.32.3 Notes for bib-style writers

The `.bib` files include records in the format:

```
@⟨entry-type⟩{⟨label⟩,
   ⟨field-name⟩ = "⟨field-data⟩",
   ⟨field-name⟩ = "⟨field-data⟩",
   ...etc
}
```

see the file `demo/op-biblist.bib` for a real example. The ⟨*entry-types*⟩ and ⟨*field-names*⟩ are case insensitive. More field-names can behave equally if the `\_fieldalias{`⟨*new-field-name*⟩`}{`⟨*given-field-name*⟩`}` is used in a style file. If a ⟨*new-field-name*⟩ is declared by this command and it is used in the `.bib` file then the effect is the same as if it was used the ⟨*given-field-name*⟩.

Ancient BibTEX has read such files and has generated files appropriate for reading by LATEX. It has worked with a set of ⟨*entry-types*⟩, see the www page `http://en.wikipedia.org/wiki/BibTeX`. The set of entry types listed on this www page is de facto the BibTEX standard. The OpTEX bib style writer must “declare” all such entry types and more non-standard entry types can be declared too if there is a good reason for doing it. The word “declare” used in the previous sentence means that a bib-style writer must define the printing rules for each ⟨*entry-type*⟩. The printing rules for ⟨*entry-type*⟩ include: which fields will be printed, in what order, by what format they will be printed on (italic, caps, etc.), which fields are mandatory, which are optional, and which are ignored in `.bib` records.

The style writer can be inspired by two styles already done: `bib-simple.opm` and `bib-iso690.opm`. The second one is documented in detail in section 2.32.6.

The printing rules for each ⟨*entry-type*⟩ must be declared by `\_sdef{_print:`⟨*entry-type*⟩`}` in `bib-`⟨*style*⟩`.opm` file. The ⟨*entry-type*⟩ has to be lowercase here. OpTEX supports following macros for a more comfortable setting of printing rules:

- `\_bprinta [`⟨*field-name*⟩`] {`⟨*if defined*⟩`} {`⟨*if not defined*⟩`}`. The part ⟨*if defined*⟩ is executed if ⟨*field-name*⟩ is declared in .bib file for the entry which is currently processed. Else the part ⟨*if not defined*⟩ is processed. The part ⟨*if defined*⟩ can include the `*` parameter which is replaced by the value of the ⟨*field-name*⟩.
- The part ⟨*if not defined*⟩ can include the `\_bibwarning` command if the ⟨*field-name*⟩ is mandatory.
- `\_bprintb [`⟨*field-name*⟩`] {`⟨*if defined*⟩`} {`⟨*if not defined*⟩`}`. The same as `\_bprinta`, but the `##1` parameter is used instead `*`. Differences: `##1` parameter can be used more than once and can be enclosed in nested braces. The `*` parameter can be used at most once and cannot be enclosed in braces. Warning: if the `\_bprintb` commands are nested (`\_bprintb` in `\_bprintb`), then you need to write the `####1` parameter for internal `\_bprintb`. But if `\_bprinta` commands are nested then the parameter is not duplicated.
- `\_bprintc \macro {`⟨*if non-empty*⟩`}`. The ⟨*if non-empty*⟩ part is executed if `\macro` is non-empty. The `*` parameter can be used, it is replaced by the `\macro`.
- `\_bprintv [`⟨*field1*⟩`,`⟨*field2*⟩`,...] {`⟨*if defined*⟩`} {`⟨*if not defined*⟩`}`. The part ⟨*if defined*⟩ is executed if ⟨*field1*⟩ or ⟨*filed2*⟩ 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 ⟨*entry-type*⟩. The formatting does not solve the numbering and paragraph indentation of the entry. This is processed by `\_printbib` macro used in OpTEX (and may be redefined by the author or document designer).

The `\bibmark={something}` can be declared, for instance in the `_print:END` macro. Such “bibmark” is saved to the `.ref` file and used in next TEX run as `\cite` marks when `\nonumcitations` is set.

Moreover, the bib-style writer must declare the format of special fields `author` and `editor`. These fields include a list of names, each name is processed individually in a loop. The `\_authorname` or `\_editorname` is called for each name on the list. The bib-style writer must define the `\_authorname` and `\_editorname` commands in order to declare the format of printing each individual name. The following control sequences can be used in these macros:

- `\_NameCount`: the number of the currently processed author in the list
- `\_namecount`: the total number of the authors in the list
- `\_Lastname`, `\_Firstname`, `\_Von`, `\_Junior`: the parts of the name.

The whole style file is read in the group during the `\usebib` command is executed before typesetting the reference list. Each definition or setting is local here.

The auto-generated phrases (dependent on current language) can be used in bib-style files by `\_mtext{bib.`⟨*identifier*⟩`}`, where ⟨*identifier*⟩ is an identifier of the phrase and the phrase itself is defined by `\_sdef{_mt:bib.`⟨*identifier*⟩`:`⟨*language*⟩`}{`⟨*phrase*⟩`}`. See section 2.37.2 for more detail. Phrases for ⟨*identifiers*⟩: and, etal, edition, citedate, volume, number, prepages, postpages, editor, editors, available, availablealso, bachthesis, masthesis, phdthesis are defined already, see the end of section 2.37.2.

The `sortedby` field is declared by `\readbibs` as a special field where sorting phrase can be specified. If it is present then it has precedence before default sorting phrase generated by `\_preparebibsorting` from the lastname, firstnames of the first author and from the year. Suppose that the .bib file includes:

```
author   = "Jan Chadima",
sortedby = "Hzzadima Jan",
```

Now, this author is sorted between H and I, because the Ch digraph in this name has to be sorted by this rule.

If you need (for example) to place the auto-citations before other citations, then you can mark your entries in `.bib` file by `sortedby = "@"`, because this character is sorted before `A`.

If you want to declare a different sorting rule, you can re-define the `\_preparebibsorting` macro. The example is in the OpTEX trick 0113.

2.32.4 Direct reading of `.bib` files

`\readbibs` `{`⟨*bib-bases*⟩`}` is internally used (by `\usebib`) for reading `.bib` databases in BibTEX format. The ⟨*bib-bases*⟩ is comma separated list of file names (without `.bib` extension, without spaces). These files are read and `\readbibs` defines macros `\_be:`⟨*bibpart*⟩`/`⟨*label*⟩, where ⟨*label*⟩ is the label of the reference record. These macros include key-value pairs `[`⟨*field name*⟩`]{`⟨*field data*⟩`}`. The first pair is `[@]{`⟨*entry type*⟩`}`. For example, if we have in the `.bib` file:

```
@Book { tbn,
   author = "Petr Olšák",
   TITle  = {\TeX{}book naruby},
   publisher = "Konvoj",
   year      = 2001,
}
```

and the `\bibpart` is empty (default value) then the `\_be:/tbn` macro is defined with the content:

```
[@]{BOOK}[author]{{Olšák}{Petr}{}{}}[authornum]{1}[title]{\TeX{}book naruby}%
[publisher]{Konvoj}[year]{2001}
```

If you do `\slet{tmp}{_be:/tbn}` then you can print the data (for example) by:

```
\ea\foreach \tmp \do [#1]#2{\wterm{field-name: "#1", data: "#2"}}
```

or you can do `\ea\foreach \tmp \do [#1]#2{\sdef{bib-field:#1}{#2}}` to enable direct access to the scanned data.

Note that entry type and field names are converted lower-case by the `\readbibs` macro.

There are two special entry types: `@COMMENT{`⟨*ignored text*⟩`}` and `@TEXCODE{`⟨*processed text*⟩`}`. The ⟨*ignored text*⟩ is ignored, the ⟨*processed text*⟩ is executed by TEX. The definitions of macros used in other entries in data of fields can be here. If the `\usebib` is used then the ⟨*processed text*⟩ is executed inside a TEX group, so the assignment is locally valid only during creating the reference list. The BiBTEX's `@STRING{}` isn't supported. All others entry types are interpreted as a *reference entry* and they are interpreted as described above. An optional balanced text between entries in `.bib` files is ignored.

If the macro `\_be:`⟨*bibpart*⟩`/*` is defined then the `\readbibs` macro reads all entries from `.bib` files and creates `\_citelist`. If the `\_be:`⟨*bibpart*⟩`/*` is undefined then the `\readbibs` macro reads only entries with ⟨*label*⟩ where `\_be:`⟨*bibpart*⟩`/`⟨*label*⟩ is set to the empty macro. After reading, the macros `\_be:`⟨*bibpart*⟩`/`⟨*label*⟩ are globally re-defined as described above.

The `\readbibs` macro doesn't convert fields data, but there are two exceptions: author and editor fields. These fields have very specific format with various alternatives, see `https://nwalsh.com/tex/texhelp/bibtx-23.html`. Shortly speaking, more authors are divided by the `and` keyword and names of a single author must be separated to four subfields: ⟨*Lastnames*⟩, ⟨*Firstnames*⟩, ⟨*Von*⟩, ⟨*Junior*⟩. Only the ⟨*Lastnames*⟩ subfield must be nonempty. The input can look like

```
Leonardo Piero da Vinci
or
da Vinci, Leonardo Piero
```

and both these variants are converted to `{Vinci}{Leonardo Piero}{da}{}`. The ⟨*Von*⟩ part is rekognized as a word with only lowercase letters. In general, the name can be written without commas: ⟨*Firstnames*⟩ ⟨*Von*⟩ ⟨*Lastnames*⟩ or with single comma: ⟨*Von*⟩ ⟨*Lastnames*⟩, ⟨*Firstnames*⟩ or with two commas: ⟨*Von*⟩ ⟨*Lastnames*⟩, ⟨*Junior*⟩, ⟨*Firstnames*⟩ and all these variants are converted to the quaternion `{`⟨*Lastnames*⟩`}{`⟨*Firstnames*⟩`}{`⟨*Von*⟩`}{`⟨*Junior*⟩`}` by the `\readbibs` macro. If there are more than single author, then each author is saved in four subfields side by side, so you have 4 or 8 or 12 etc. subfildeds in the author/editor data field. You can read them by `\foreach` ⟨*author-data*⟩`\do #1#2#3#4{...}`.

2.32.5 The `usebib.opm` macro file loaded when `\usebib` is used

usebib.opm

```
\_codedecl \readbibs {Reading bib databases <2024-02-18>} % loaded on demand by \usebib
```

First, we implement the scanner of `.bib` files. Unfortunately, the format of these files isn't TEX friendly, so we must to do more work. `\readbibs` `{`⟨*bib-bases*⟩`}` reads ⟨*bib-bases*⟩ files (i.e. the BibTEX format).

usebib.opm

```
\_newcount\_aunum
\_newcount\_NameCount
\_def\_eaddto#1#2{\_ea\_addto\_ea#1\_ea{#2}}

\_def\_readbibs #1{%
   \_ifcsname _be:\_bibp*\_endcsname \_def\_citelist{}\_fi % \_citelist will be created
   \_begingroup
      \_everyeof{@{}}\_foreach#1,\_do##1,{%
          \_isfile{##1.bib}\_iftrue \_ea\_nextat\_input{##1.bib}
          \_else \_opwarning{\_string\usebib: Missing ##1.bib file}
          \_fi}%
   \_endgroup
}
\_public \readbibs ;
```

The `\_nextat` macro skips the text in the `.bib` file to the next `@`, and starts the `\_bibentry` macro which reads `@`⟨*entry type*⟩`{`⟨*data*⟩`}` from the `.bib` file. Each reference entry is converted to the `\_entrydata` macro and then `\_glet \_be:`⟨*bib-part*⟩`/`⟨*label*⟩ `= \_entrydata` is done. The `\_entrydata` includes key-value pairs, as described in the section 2.32.4.

usebib.opm

```
\_long\_def\_nextat#1@{\_bgroup\_catcode` =9 \_ea\_egroup\_bibentry}
\_def\_bibentry #1#{\_ifx^#1^\_else \_afterfi{\_bibentryA{#1}}\_fi}
\_def\_bibentryA #1#2{\_lowercase{\_def\_entrytype{#1}}%
   \_ismacro\_entrytype{comment}\_iffalse  % comment is ignored
   \_ismacro\_entrytype{texcode}\_iftrue   % TeX code is processed
      \_endgroup #2\_begingroup \_everyeof{@{}}\_else
   \_ismacro\_entrytype{string}\_iftrue    % string is reported as unsupported
      \_opwarning{\_string\usebib: @STRING entry isn't supported, try to use @TEXCODE}%
   \_else
      \_edef\_entrydata{[@]{\_entrytype}}%
      {\_bibentryB #2\_fin}%               % read a "normal" bib entry
   \_fi\_fi\_fi
   \_nextat
}
\_def\_bibentryB #1#2,#3\_fin{\_def\_citekey{#1#2}\_def\_bibentryC{\_nextfield #3,\_fin}%
   \_ifcsname _be:\_bibp*\_endcsname
      \_bibentryC
      \_global\_addto\_citelist{\_citeI[#1#2]}
   \_else
   \_ea\_ifx \_begincsname _be:\_bibp#1#2\_endcsname \_empty
      \_bibentryC
   \_fi\_fi
}
\_def\_bibentryF {% finalize entry
   \_preparebibsorting
   \_global\_ea\_let \_csname _be:\_bibp\_citekey\_endcsname = \_entrydata
}
```

`\_nextfield` reads next field name and saves it to the `\_fieldname` and then reads field data and saves it to the `\_fielddata`.

usebib.opm

```
\_def\_nextfield #1{\_ifx,#1\_ea\_nextfield   % skip commas from previous field
   \_else \_ifx\_fin#1\_ea\_ea\_ea\_bibentryF % finalize bib entry
   \_else \_def\_fieldname{#1}\_ea\_ea\_ea\_nextfieldA % first letter of fieldname found
   \_fi\_fi
}
\_def\_nextfieldA #1{% next letters of field name until = is found
   \_ifx=#1\_afterfi{\_nospacefuturelet\_next\_nextfieldB}%
   \_else \_addto\_fieldname{#1}%
   \_ea\_nextfieldA \_fi
}
\_def\_nextfieldB {% reading field data
   \_casesof\_next
   "        {\_nextfieldC}%  name = "data",
   \_bgroup {\_nextfieldD}%  name = {data},
   \_finc   {\_nextfieldE}%  name = data,
}
\_def\_nextfieldC "#1"{\_nextfieldD{#1}}
\_def\_nextfieldD #1{\_def\_fielddata{#1}\_nextfieldF}
```

```
\_def\_nextfieldE #1,{\_nextfieldD{#1}}

\_def\_nextfieldF{% finalize field
   \_lowercase\_ea{\_ea\_def\_ea\_fieldname\_ea{\_fieldname}}% case insensitive field name
   \_ifcsname _fia:\_fieldname\_endcsname \_edef\_fieldname{\_cs{_fia:\_fieldname}}\_fi
   \_eaddto\_entrydata{\_ea[\_fieldname]}%
   \_ismacro\_fieldname{author}\_iftrue \_ea\_auscan\_ea{\_fielddata}[author]\_else
   \_ismacro\_fieldname{editor}\_iftrue \_ea\_auscan\_ea{\_fielddata}[editor]\_else
   \_eaddto\_entrydata{\_ea{\_fielddata}}\_fi\_fi
   \_sdef{_fd:\_fieldname\_ea}\_ea{\_fielddata}%
   \_nextfield
}
```

`\_fieldalias`{⟨*new-name*⟩}{⟨*given-name*⟩} defines `\_fia:`⟨*new_name*⟩ as ⟨*given-name*⟩.

usebib.opm

```
\_def\_fieldalias#1#2{\_lowercase{\_sxdef{_fia:#1}{#2}}}
```

The `\_auscan`{⟨*authors/editors-names*⟩}[⟨*field-name*⟩] reads the specific BibTEX format menitoned in section 2.32.4 and converts them to {⟨*Lastname*⟩}{⟨*Firstname*⟩}{⟨*Von*⟩}{⟨*Junior*⟩} for each author/editor. The result includes $4k$ subfields (where k is number of the authors/editors) and it is saved to `\_entrydata` and the `[authornum]{`k`}` or `[editornum]{`k`}` is added.

The `\_auscanA` macro does the loop over authors separated by `and`. Each single author has its `\_tmpb` macro with `X` and `x`. Each letter corresponds to single word of the name (`X`: begins with uppercase, `x`: begins with lowercase). For example Leonardo Piero da Vinci has `\_tmpb` macro `XXxX.`. If there are commas in after some words, then these commas are in `\_tmpb` macro too, for example da Vinci, Piero Leonardo has its `\_tmpb` macro `xX,XX.`. The number of commas is saved to `\_tmpnum`. The `\_auscanB` macro does a slight modifications of the `\_tmpb` macro as mentioned in comments. Then the macro `\_auscanD`⟨*tpmb-pattern*⟩`;{`⟨*WordA*⟩`}{`⟨*WordB*⟩`}{`⟨*WordC*⟩`}...` is executed. It saves given words due to the `\_tmpb` pattern to the macros `\_Lastname`, `\_Firstname`, `\_Von`, `\_Junior` in a loop. Finally, the contents of these macros are saved to `\_fiedldata` and then to the `\_entrydata`.

usebib.opm

```
\_def\_auscan#1[#2]{\_def\_auname{}\_def\_fielddata{}\_def\_tmpb{}\_aunum=0\_tmpnum=0
   \_auscanA #1 and {}
   \_eaddto\_entrydata{\_ea{\_fielddata}[#2num]}%
   \_eaddto\_entrydata{\_ea{\_the\_aunum}}%
}
\_def\_auscanA #1 {%
   \_ifx^#1^\_else
      \_isequal{#1}{and}\_iftrue
         \_incr\_aunum
         \_addto\_tmpb{.}%
         \_auscanB
         \_ea\_auscanX \_auname
         \_def\_auname{}\_def\_tmpb{}\_tmpnum=0
      \_else
         \_lowercase{\_isequal{#1}}{#1}\_iftrue \_addto\_tmpb{x}\_else \_addto\_tmpb{X}\_fi
         \_def\_tmp{#1^}\_isinlist\_tmp{,^}%
         \_iftrue
             \_ea\_auscanC\_tmp \_addto\_tmpb{,}\_incr\_tmpnum
         \_else
             \_def\_tmp{#1}%
         \_fi
         \_eaddto\_auname{\_ea{\_tmp}}%
      \_fi
      \_ea\_auscanA
   \_fi
}
\_def\_auscanB{%
   \_ifcase\_tmpnum  % 0 commas:  XXX. -> XX:X. ; XxxXX. -> X:xxXX. ;  xXX. -> :xXX. ; First:Last
      \_isinlist\_tmpb{x}\_iffalse \_replstring\_tmpb{X.}{:X.}\_else
      \_ea\_auscanT\_tmpb;\_iffalse \_replstring\_tmpb{Xx}{X:x}\_fi\_fi
   \_or             % 1 comma:  XX,XXX -> XX,,XXX, Junior part is empty
      \_replstring\_tmpb{,}{,,}\_tmpnum=2
   \_fi             % 2 commas: XX,XX,XXX no changes, generic format: Last, Junior, First
   \_def\_Firstname{}\_def\_Lastname{}\_def\_Von{}\_def\_Junior{}%
}
\_def\_auscanT #1#2;\_iffalse{\_ifx#1x\_def\_tmpb{:x#2}\_else} % xXX. -> :xXX.
```

```
\_def\_auscanX {\_ea\_auscanD\_tmpb;}
\_def\_auscanD #1#2;{%
   \_def\_tmpb{#2}%
   \_casesof #1
   . {\_auscanF}                              % dot is last character, do final job
   , {\_decr\_tmpnum \_auscanX}                % Lastname->Junior or Junior->Firstname
   : {\_tmpnum=2 \_auscanX}                   % Firstname->Lastname
   X {\_auscanE\_Firstname\_Junior\_Lastname} % add data due to the \tmpnum value
   x {\_auscanE\_Firstname\_Junior\_Von}      % Von instead Lastname
   \_finc {}%
}
\_def\_auscanE#1#2#3{\_ifcase\_tmpnum \_ea\_auscanS\_ea#1\_or \_ea\_auscanS\_ea#2\_else
                                  \_ea\_auscanS\_ea#3\_fi}
\_def\_auscanS#1#2{% #1=\Firstname or \Lastname or etc., #2=word to be inserted
   \_ifx#1\_empty \_def#1{#2}\_else \_addto#1{ #2}\_fi
   \_auscanX
}
\_def\_auscanF{% final work of \auscanX
   \_eaddto\_fielddata{\_ea{\_Lastname}}\_eaddto\_fielddata{\_ea{\_Firstname}}%
   \_eaddto\_fielddata{\_ea{\_Von}}\_eaddto\_fielddata{\_ea{\_Junior}}%
}
\_def\_auscanC #1,^{\_def\_tmp{#1}}  % removing final comma: Word,^ -> Word
```

The `\_citelist` includes `\_citeI`[⟨*label*⟩] commands. The `\usebib` macro runs this lists in order to print references. Each `\_citeI`[⟨*label*⟩] prints single bib entry given by the ⟨*label*⟩. It opens a group, sets macros `\_fd:`⟨*field-name*⟩ to ⟨*field-data*⟩ and runs `\_printentry`. Finally, it closes TeX group, so all macros `\_fd:`⟨*field-name*⟩ have their initial (undefined) value.
The `\_getfield`[⟨*field-name*⟩]`\macro` does `\def\macro{`⟨*field-data*⟩`}`. If the field isn't declared then the `\macro` is empty.

usebib.opm

```
\_def\_citeI[#1]{%
   \_begingroup
      \_ea\_ifx \_begincsname _be:\_bibp#1\_endcsname \_empty
         \_opwarning{\_string\usebib: entry [#1] isn't found in .bib}%
         \_global\_slet{_bes:#1}{_relax}%
      \_else
         \_ea\_ea\_ea\_foreach \_csname _be:\_bibp#1\_endcsname \_do[##1]##2{\_sdef{_fd:##1}{##2}}%
         \_def\_entrykey{#1}%
         \_printentry
      \_fi
   \_endgroup
}
\_def\_getfield[#1]#2{%
   \_ifcsname _fd:#1\_endcsname
       \_ea\_ea\_ea \_def \_ea\_ea\_ea #2\_ea\_ea\_ea {\_csname _fd:#1\_endcsname}%
   \_else \_def#2{}%
   \_fi
}
```

`\_preparebibsorting` is called repeatedly for each bib entry when its reading from `.bib` file is finished. Its main goal is to do `\gdef\_bes:`⟨*citekey*⟩ `{\;`⟨*sorting-rule*⟩`^^^`⟨*citekey*⟩`}`. Note that the part of the control sequence name after `^^^` is ignored during sorting. The default `\_preparebibsorting` macro creates ⟨*sorting-rule*⟩ in the form: ⟨*Lastnames*⟩ ⟨*Firstnames*⟩ ⟨*Von*⟩ ⟨*Junior*⟩ of the first author followed by ⟨*year*⟩ from `year` field.
`\_dobibsorting\_citelist` sorts the `\_citelist` and runs it.

usebib.opm

```
\_def\_preparebibsorting{%
   \_getfield[sortedby]\_sortedby
   \_ifx\_sortedby\_empty   % explicitly given [sortedby] field has precedence
      \_def\_sortedby{}%
      \_getfield[author]\_tmp  % sorting by author firstly
      \_ifx\_tmp\_empty \_else \_ea\_preparebibsortingA\_tmp \_fin \_fi
      \_getfield[year]\_tmp    % soering by year secondly
      \_ifx\_tmp\_empty \_else \_eaddto\_sortedby{\_tmp}\_fi
      \_edef\_sortedby{\_sortedby}%  we need to run macros aka e etc.
      \_edef\_sortedby{\_ea\_removeoutbraces\_sortedby {\_fin}}% remove braces
   \_fi
   \_sxdef{_bes:\_citekey\_ea}\_ea{\_csname;\_detokenize\_ea{\_sortedby}^^^\_citekey\_endcsname}%
}
\_def\_preparebibsortingA#1#2#3#4#5\_fin {% names of the first author used by sorting:
   \_def\_sortedby{#1 }%                        Lastname
   \_ifx^#2^\_else \_addto\_sortedby{#2 }\_fi  % Firstname
   \_ifx^#3^\_else \_addto\_sortedby{#3 }\_fi  % Von
   \_ifx^#4^\_else \_addto\_sortedby{#4 }\_fi  % Junior
}
\_def\_dobibsorting{%
   {\_def\_citeI[##1]{\_ea\_citeIs\_csname _bes:##1\_endcsname{##1}}%
    \_edef\_citelist{\_ea}\_citelist % converting \_citelist
    \_dosorting\_citelist \_ea}%
}
\_def\_citeIs#1#2{\_eaddto\_citelist{#1}%
   \_ea\_gdef#1{\_citeI[#2]\_ea\_glet#1=\_undefined \_glet#1=\_undefined}%
}
```

The `\_printentry` macro prints bibliographic reference entry. It prints ⟨*bibnum*⟩ or ⟨*bimark*⟩ (including hyperlinks) and they are followed by printing the entry data. The format is given by the `\_printbib` macro and by `\_print:`⟨*entrytype*⟩ declared in the bib-style file.

usebib.opm

```
\_def\_printentry {\_par \_bibskip
  \_bibgl\_incr\_bibnum
  \_isdefined{_bim:\_bibp\_the\_bibnum}\_iftrue
     \_edef\_tmpb{\_csname _bim:\_bibp\_the\_bibnum\_endcsname}%
     \_bibmark=\_ea{\_tmpb}%
  \_else \_bibmark={}\_fi
  \_edef\_tmpb{\_entrykey}%
  \_noindent \_dbib\_entrykey
  \_printbib
  {%
     \_getfield[@]\_entrytype
     \_csname _print:BEGIN\_endcsname
     \_isdefined{_print:\_entrytype}\_iftrue
        \_csname _print:\_entrytype\_endcsname
     \_else
        \_ifx\_entrytype\_empty \_else
           \_opwarning{Entrytype @\_entrytype\_space from [\_entrykey] undefined}%
           \_csname _print:misc\_endcsname
     \_fi\_fi
     \_csname _print:END\_endcsname
     \_wbib \_entrykey {\_the\_bibnum}{\_the\_bibmark}%
  }\_par
}
```

The `\_bprinta`, `\_bprintb`, `\_bprintc`, `\_bprintv` commands used in the style files:

usebib.opm

```
\_def\_bprinta {\_bprintb*}
\_def\_bprintb #1[#2#3]{%
   \_def\_bibfieldname{#2#3}%
   \_if!#2\_relax
      \_def\_bibfieldname{#3}%
      \_getfield[#3]\_bibfield
      \_getfield[#3num]\_namecount % total persons in the author/editor fields
      \_ifx\_bibfield\_empty\_else
         \_def\_bibfield{\_loopauthors{#3}}% read author/editor field in a loop
      \_fi
   \_else
      \_getfield[#2#3]\_bibfield
   \_fi
   \_if^#1^%
      \_ifx\_bibfield\_empty \_ea\_ea\_ea \_doemptyfield
      \_else \_ea\_ea\_ea \_dofullfield \_fi
   \_else \_ea \_bprintaA
   \_fi
}
\_def\_dofullfield#1#2{\_def\_dofield##1{#1}\_ea\_dofield\_ea{\_bibfield}}
\_def\_doemptyfield#1#2{\_def\_dofield##1{#2}\_ea\_dofield\_ea{\_bibfield}}
\_def\_bprintaA #1#2{\_ifx\_bibfield\_empty #2\_else\_bprintaB #1**\_fin\_fi}
```

```
\_def\_bprintaB #1*#2*#3\_fin{\_ifx^#3^#1\_else\_ea\_bprintaC\_ea{\_bibfield}{#1}{#2}\_fi}
\_def\_bprintaC #1#2#3{#2#1#3}
\_def\_bprintc#1#2{\_bprintcA#1#2**\_relax}
\_def\_bprintcA#1#2*#3*#4\_relax{\_ifx#1\_empty \_else \_if^#4^#2\_else#2#1#3\_fi\_fi}
\_def\_bprintv [#1]#2#3{\_def\_tmpa{#2}\_def\_tmpb{#3}\_bprintvA #1,,}
\_def\_bprintvA #1,{%
   \_if^#1^\_tmpb\_else
      \_getfield[#1]\_tmp
      \_ifx \_tmp\_empty
      \_else \_tmpa \_def\_tmpb{}\_def\_tmpa{}%
      \_fi
   \_ea \_bprintvA
   \_fi
}
```

`\_loopauthors{`⟨*field-name*⟩`}` does a loop over all authors/editors in the `author` or `editor` field. The `\_namecount` (total number of authors/editors) was defined in `\_bprintb`. Then for each author/edtor it do:

- Set `\_NameCount` to the position number of the currently processed author/editor.
- Define `\_Lastname`, `\_Firstname`, `\_Junior`, `\_Von`, `\_After` macros.
- Run `\_authorname` or `\_editorname` macro (defined in the bib style file).

usebib.opm

```
\_def\_loopauthors #1{%
   \_NameCount=0
   \_ea\_ea\_ea\_foreach\_csname _fd:#1\_endcsname \_do ##1##2##3##4{%
      \_advance\_NameCount by1
      \_def\_Lastname{##1}\_def\_Firstname{##2}\_def\_Von{##3}\_def\_Junior{##4}%
      \_csname _#1ini\_endcsname \_csname _#1name\_endcsname
}}
\_def\_authorini{} % ready for \_AbbreviateFirstname or similar...
\_def\_editorini{}
```

`\_bibwarning` can be used if the manatory field is missing. Note that `\_nobibwarnlist` is used here, it is set by `\nobibwarning` macro.

usebib.opm

```
\_def\_bibwarning{%
   \_ea\_isinlist \_ea\_nobibwarnlist\_ea{\_ea,\_entrykey,}\_iffalse
       \_opwarning{Missing field "\_bibfieldname" in [\_entrykey]}\_fi}
```

`\_AbbreviateFirstname`, `\_RetrieveFieldIn`, `\_RetrieveField` are here only for backward compatibility with previous macros based on the librarian package. The `\_CreateField`, `\_SortingOrder`, and `\_SpecialSort` are dummy macros because the sorting is implemented by a slightly different way than in librarian package.

usebib.opm

```
\_def\_AbbreviateFirstname{\_addto\_authorini{\_abbrevnames\_Firstname}}
\_def\_abbrevnames#1{% Karolina Pondelickova-Maslova -> K. P.-M.
   \_edef#1{\_ea\_foreach #1 \_do ##1##2 {##1.%
                \_foreach ##2-{}\_do ####1-####2{\_ifx^####2^\_else-####2.\_fi} }^%
   }\_replstring#1{ ^}{}%
}
\_def\_RetrieveFieldIn#1{\_getfield[#1]}
\_def\_RetrieveField#1{\_trycs{_fd:#1}{}}
\_def\_CreateField#1{}
\_def\_SortingOrder#1#2{}
\_def\_SpecialSort#1{}
```

The `\usebib` command is defined as `\input{usebib.opm}\_usebib` in the format. So, the command is re-defined here and it is run again with the new meaning.
The `\usebib` macro defined here reads `\_ctlst:`⟨*bibpart*⟩`/` and `\_ctlstB:`⟨*bibpart*⟩`/` (they include a list of `\_citeI[`⟨*label*⟩`]`) and merges them to a single `\_citelist`. The `\_be:`⟨*bibpart*⟩`/`⟨*label*⟩ is set to empty for each member of the `\_citelist`. Then the style file is read in a group, the `\readbibs` macro reads given `.bib` files and resulting `\_citelist` is processed: i.e. the macros `\_citeI` print desired entries.

usebib.opm

```
\_def\_usebib/#1 (#2) #3 {%
   \_ifcsname _ctlst:\_bibp\_endcsname
       \_slet{_citelist}{_ctlst:\_bibp}\_else \_def\_citelist{}\_fi
   \_ea \_foreach\_citelist \_do ##1[##2]{\_sdef{_be:\_bibp##2}{}}%
   \_ifcsname _ctlstB:\_bibp\_endcsname
     \_ea\_ea\_ea\_foreach \_csname _ctlstB:\_bibp\_endcsname \_do ##1[##2]{%
         \_ifcsname _be:\_bibp##2\_endcsname
         \_else \_addto\_citelist{\_citeI[##2]}\_sdef{_be:\_bibp##2}{}%
         \_fi
     }%
   \_fi
   \_global \_ea\_let \_csname _ctlst:\_bibp\_endcsname =\_write
   \_ifx\_citelist\_empty
     \_opwarning{No cited items. \_noexpand\usebib ignored}%
   \_else
     \_bgroup
        \_par
        \_emergencystretch=.3\_hsize
        \_def\_optexbibstyle{#2}%
        \_setctable\_optexcatcodes
        \_input bib-#2.opm
        \_the \_bibtexhook
        \_ifcsname _mt:bib.and:\_cs{_lan:\_the\_language}\_endcsname \_else
           \_opwarning{\_string\usebib: No phrases for language
                       "\_cs{_lan:\_the\_language}" (using "en")}%
           \_language=0 \_chardef\_documentlanguage=0
        \_fi
        \_ifx#1c\_def\_preparebibsorting{}\_def\_dobibsorting{}\_fi
        \_readbibs {#3}%
        \_dobibsorting\_citelist
        \_restorectable
     \_egroup
   \_fi
}
```

2.32.6 Usage of the `bib-iso690` style

This is the iso690 bibliographic style used by OpTEX.

See `op-biblist.bib` for an example of the `.bib` input. You can try it by:

```
\fontfam[LMfonts]
\nocite[*]
\usebib/s (iso690) op-biblist
\end
```

Common rules in `.bib` files

There are entries of type `@FOO{...}` in the `.bib` file. Each entry consists of fields in the form `name␣=␣"value"`, or `name␣=␣{value}`. No matter which form is used. If the value is pure numeric then you can say simply `name␣=␣value`. Warning: the comma after each field value is mandatory! If it is missing then the next field is ignored or badly interpreted.

The entry names and field names are case insensitive. If there exists a data field no mentioned here then it is simply ignored. You can use it to store more information (abstract, for example).

There are "standard fields" used in ancient 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.

```
author = "John Green and Bob Brown and Alice Black",
option = "auetal autrim:1",
```

output: GREEN, J., B. BROWN, A. BLACK et al.

If you need to write a team name or institution instead of authors, replace all spaces by `\␣` in this name. Such text is interpreted as Lastname. You can add the secondary name (interpreted as Firstname) after the comma. Example:

```
author = "Czech\ Technical\ University\ in\ Prague,
          Faculty\ of\ Electrical\ Engineering",
```

output: CZECH TECHNICAL UNIVERSITY IN PRAGUE, Faculty of Electrical Engineering.

The editor field

The editor field is used for the list of the authors of the collection. The analogous rules as in author field are used here. It means that the authors are separated by "` and `", the Firstnames, Lastnames, etc. are interpreted and you can use the options `edmax:`⟨*number*⟩, `edmin:`⟨*number*⟩, `edetal`, `edtrim:`⟨*number*⟩ and `edprint:{`⟨*value*⟩`}` (with `\ED` macro). Example:

```
editor = "Jan Tomek and Petr Karas",
option = "edprint:{\ED, editors.} edtrim:1",
```

Output: J. TOMEK and P. KARAS, editors.

If `edprint` option is not set then `{\ED,␣eds.}` or `{\ED,␣ed.}` is used depending on the entry language and on the singular or plural of the editor(s).

The ednote field

The ednote field is used as the secondary authors and more editional info. The value is read as raw data without any interpretation of Lastname, Firstname etc.

```
ednote = "Illustrations by Robert \upper{Agarwal}, edited by Tom \upper{Nowak}",
```

output: Illustrations by Robert AGARWAL, edited by Tom NOWAK.

The `\upper` command has to be used for Lastnames in the ednote field.

The title field

This is the title of the work. It will be printed (in common entry types) by italics. The ISO 690 norm declares, that the title plus optional subtitle are in italics and they are separated by a colon. Next, the optional secondary title has to be printed in an upright font. This can be added by `titlepost:{`⟨*value*⟩`}`. Example:

```
title = "The Simple Title of The Work",
or
title = "Main Title: Subtitle",
or
title  = "Main Title: Subtitle",
option = "titlepost:{Secondary title}",
```

The output of the last example: *Main Title: Subtitle.* Secondary title.

The edition field

This field is used only for second or more edition of cited work. Write only the number without the word "edition". The shortcut "ed." (or something else depending on the current language) is added automatically. Examples:

```
edition = "Second",
edition = "2nd",
edition = "2$^{\rm nd}$",
edition = "2.",
```

Output of the last example: 2. ed.

```
edition = "2."
lang    = "cs",
```

Output: 2. vyd.

Note, that the example `edition␣=␣"Second"` may cause problems. If you are using language ”cs” then the output is bad: Second vyd. But you can use `editionprint:{`⟨*value*⟩`}` option. The the ⟨*value*⟩ is printed instead of edition field and shortcut. The edition field must be set. Example:

```
edition = "whatever",
option  = "editionprint:{Second full revised edition}",
```

Output: Second full revised edition.

You can use `\EDN` macro in `editionprint` value. This macro is expanded to the edition value. Example:

```
edition = "Second",
option  = "editionprint:{\EDN\space full revised edition}",
or
edition = "Second full revised edition",
option  = "editionprint:{\EDN}",
```

The address, publisher, year fields

This is an anachronism from ancient BibTEX (unfortunately no exclusive) that the address field includes only the city of the publisher's residence. No more data are here. The publisher field includes the name of the publisher.

```
address = "Berlin",
publisher = "Springer Verlag",
year = 2012,
```

Output: Berlin: Springer Verlag, 2012.

Note, that the year needn't to be inserted into quotes because it is pure numeric.

The letter a, b, etc. are appended to the year automatically if two or more subsequent entries in the bibliography list are not distinct by the first author and year fields. If you needn't this feature, you can use the `noautoletters` option.

You can use ”yearprint:⟨*value*⟩” option. If it is set then the ⟨*value*⟩ is used for printing year instead the real field value. The reason: year is sort sensitive, maybe you need to print something else than only sorting key. Example:

```
year   = 2000,
option = "yearprint:{© 2000}",
```

Output: © 2000, sorted by: 2000.

```
year   = "2012a",
option = "yearprint:{2012}",
```

Output: 2012, sorted by: 2012a.

The address, publisher, and year are typically mandatory fields. If they are missing then the warning occurs. But you can set `unpublished` option. Then this warning is suppressed. There is no difference in the printed output.

The url field

Use it without `\url` macro, but with `http://` prefix. Example:

```
url = "http://petr.olsak.net/opmac.html",
```

The ISO 690 norm recommends to add the text “Available from” (or something else if a different current language is used) before URL. It means, that the output of the previous example is:

Available from `http://petr.olsak.net/opmac.html`.

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 in use).

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 selected language. Example:

```
citedate = "2004/05/21",
```

Output when `en` is used: [cit. 2004-05-21].
Output when `cs` is used: [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",
```

Output: Vol. 31, No. 3, pp. 37–42.

```
volume = 31,
number = 3,
pages  = "37--42",
lang   = "cs",
```

Output: ročník 31, č. 3, s. 37–42.

If you disagree with the default prefixes, you can use the numbering field. When it is set then it is used instead of volume, number, pages fields and instead of any mentioned prefixes. The numbering can include macros `\VOL`, `\NO`, `\PP`, which are expanded to the respective values of fields. Example:

```
volume    = 31,
number    = 3,
pages     = "37--42"
numbering = "Issue~\VOL/\NO, pages~\PP",
```

Output: Issue 31/3, pages 37–42

Note: The volume, numbers, and pages fields are printed without numbering filed only in the `@ARTICLE` entry. It means, that if you need to visible them in the `@INBOOK`, `@INPROCEEDINGS` etc. entries, then you must use the numbering field.

Common notes about entries

The order of the fields in the entry is irrelevant. We use the printed order in this manual. The exclamation mark (!) denotes the mandatory field. If the field is missing then a warning occurs during processing.

If the `unpublished` option is set then the fields address, publisher, year, isbn, and pages are not mandatory. If the `nowarn` option is set then no warnings about missing mandatory fields occur.

If the field is used but not mentioned in the entry documentation below then it is silently ignored.

- The `@BOOK` entry

This is used for book-like entries.

Fields: author(!), title(!), howpublished, edition, ednote, address(!), publisher(!), year(!), citedate, series, isbn(!), doi, url, note.

The ednote field here means the secondary authors (illustrator, cover design etc.).

- The `@ARTICLE` entry

  This is used for articles published in a journal.

  Fields: author(!), title(!), journal(!), howpublished, address, publisher, month, year, [numbering or volume, number, pages(!)], citedate, issn, doi, url, note.

  If the numbering is used then it is used instead volume, number, pages.

- The `@INBOOK` entry

  This is used for the part of a book.

  Fields: author(!), title(!), booktitle(!), howpublished, edition, ednote, address(!), publisher(!), year(!), numbering, citedate, series, isbn or issn, doi, url, note.

  The author field is used for author(s) of the part, the editor field includes author(s) or editor(s) of the whole document. The pages field specifies the page range of the part. The series field can include more information about the part (chapter numbers etc.).

  The `@INPROCEEDINGS` and `@CONFERENCE` entries are equivalent to `@INBOOK` entry.

- The `@THESIS` entry

  This is used for the student's thesis.

  Fields: author(!), title(!), howpublished, address(!), school(!), month, year(!), citedate, type(!), ednote, doi, url, note.

  The type field must include the text "Master's Thesis" or something similar (depending on the language of the outer document).

  There are nearly equivalent entries: `@BACHELORSTHESIS`, `@MASTERSTHESIS` and `@PHDTHESIS`. These entries set the type field to an appropriate value automatically. The type field is optional in this case. If it is used then it has precedence before the default setting.

- The @ONLINE entry

  It is intended for online publications.

  Fields: author, title(!), howpublished, ednote, publisher, accessed, doi, url(!), note.

- The @MISC entry

  It is intended for various usage.

  Fields: author, title, howpublished, ednote, citedate, doi, url, note.

  You can use `\AU`, `\ED`, `\EDN`, `\VOL`, `\NO`, `\PP`, `\ADDR`, `\PUBL`, `\YEAR` macros in ednote field. These macros print authors list, editors list, edition, volume, number, pages, address, publisher, and year field values respectively.

  The reason for this entry is to give to you the possibility to set the format of entry by your own decision. The most of data are concentrated in the ednote field.

- The `@BOOKLET`, `@INCOLLECTION`, `@MANUAL`, `@PROCEEDINGS`, `@TECHREPORT`, `@UNPUBLISHED` entries

  These entries are equivalent to `@MICS` entry because we need to save the simplicity. They are implemented only for (almost) backward compatibility with the ancient BibTeX. But the ednote is mandatory field here, so you cannot use these entries from the old databases without warnings and without some additional work with the `.bib` file.

The cite-marks (bibmark) used when `\nonumcitations` is set

When `\nonumcitations` is set then `\cite` prints text-oriented bib-marks instead of numbers. This style file auto-generates these marks in the form "Lastname of the first author, comma, space, the year" if the bibmark field isn't declared. If you need to set an exception from this common format, then you can use bibmark field.

The OPmac trick `http://petr.olsak.net/opmac-tricks-e.html#bibmark` describes how to redefine the algorithm for bibmark auto-generating when you need the short form of the type [Au13].

Sorting

If `\usebib/c` is used then entries are sorted by citation order in the text. If `\usebib/s` is used then entries are sorted by "Lastname, Firstname(s)" of the first author and if more entries have this value equal, then the year is used (from older to newer). This feature follows the recommendation of the ISO 690 norm.

If you have the same authors and the same year, you can control the sorting by setting years like 2013, 2013a, 2013b, etc. You can print something different to the list using `yearprint{`⟨*value*⟩`}` option, see the section about address, publisher, and year above. The real value of year field (i.e. not yearprint value) is also used in the text-oriented bib-marks when `\nonumcitations` is set.

If you have some problems with name sorting, you can use the hidden field `sortedby` (or `key` field with the same effect). It can be used for sorting instead of the “Lastname Firstname(s)” of the first author. If the `sortedby` field is unset then the “Lastname Firstname(s)” is used for sorting normally. Example:

```
author    = "Světla Čmejrková",
sortedby  = "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 `sortedby␣=␣"@"`, to each entry with your name because the `@` character is sorted before `A`.

Languages

There is the language of the outer document and the languages of each entry. The ISO 690 norm recommends that the technical notes (the prefix before URL, the media type, the “and” conjunction between the semifinal and final author) maybe printed in the language of the outer document. The data of the entry have to be printed in the entry language (edition ed./vyd., Vol./ročník, No./č. etc.). Finally, there are the phrases independent of the language (for example In:). Unfortunately, the bibTEX supposes that the entry data are not fully included in the fields so the automaton has to add some text during processing (“ed.”, “Vol.”, “see also”, etc.). But what language has to be chosen?

The current value of the `\language` register at the start of the `.bib` processing is described as the language of the outer document. This language is used for technical notes regardless of the entry language. Moreover, each entry can have the `lang` field (short name of the language). This language is used for ed./vyd., vol./ročník, etc. and it is used for hyphenation too. If the `lang` is not set then the outer document language is used.

You can use `\_Mtext{bib.`⟨*identifier*⟩`}` if you want to use a phrase dependent on outer document language (no on entry language). Example:

```
howpublished = "\_Mtext{bib.blue-ray}"
```

Now, you can set the variants of `bib.blue-ray` phrase for various languages:

```
\_sdef{_mt:bib.blue-ray:en} {Blue-ray disc}
\_sdef{_mt:bib.blue-ray:cs} {Blue-ray disk}
```

Summary of non-standard fields

This style uses the following fields unknown by bibTEX:

```
option    ... options separated by spaces
lang      ... the language two-letter code of one entry
ednote    ... edition info (secondary authors etc.) or
              global data in @MISC-like entries
citedate  ... the date of the citation in year/month/day format
numbering ... format for volume, number, pages
isbn      ... ISBN
issn      ... ISSN
doi       ... DOI
url       ... URL
```

Summary of options

```
aumax:⟨number⟩         ... maximum number of printed authors
aumin:⟨number⟩         ... number of printed authors if aumax exceeds
autrim:⟨number⟩        ... full Firstnames iff number of authors are less than this
auprint:{⟨value⟩}      ... text instead authors list (\AU macro may be used)
edmax, edmin, edtrim ... similar as above for editors list
edprint:{⟨value⟩}      ... text instead editors list (\ED macro may be used)
titlepost:{⟨value⟩}    ... text after title
yearprint:{⟨value⟩}    ... text instead real year (\YEAR macro may be used)
editionprint:{⟨value⟩} .. text instead of real edition (\EDN macro may be used)
```

```
urlalso      ... the ``available also from'' is used instead ``available from''
unpublished  ... the publisher etc. fields are not mandatory
nowarn       ... no mandatory fields
```

Other options in the option field are silently ignored.

2.32.7 Implementation of the `bib-iso690` style

bib-iso690.opm

```
\_codedecl \_undefined {BIB style (iso690) <2025-04-15>} % loaded on demand by \usebib

\_ifx\_optexbibstyle\_undefined \_errmessage
   {This file can be read by: \_string\usebib/? (iso690) bibfiles command only}
   \_endinput \_fi
```

`\_maybetod` (alias `\:` in the style file group) does not put the second dot.

bib-iso690.opm

```
\_def\_maybedot{\_ifnum\_spacefactor=\_sfcode`\.\_relax\_else.\_fi}
\_tmpnum=\_sfcode`\. \_advance\_tmpnum by-2 \_sfcode`\.=\_tmpnum
\_sfcode`=\_tmpnum \_sfcode`=\_tmpnum
\_let=\_maybedot  % prevents from double periods
\_ifx\.\_undefined \_let\.=\_maybedot \_fi % for backward compatibility
```

Option field.

bib-iso690.opm

```
\_CreateField {option}
\_def\_isbiboption#1#2{\_edef\_tmp{\_noexpand\_isbiboptionA{#1}}\_tmp}
\_def\_isbiboptionA#1{\_def\_tmp##1 #1 ##2\_relax{%
      \_if^##2^\_csname iffalse\_ea\_endcsname \_else\_csname iftrue\_ea\_endcsname \_fi}%
   \_ea\_tmp\_biboptionsi #1 \_relax}
\_def\_bibopt[#1]#2#3{\_isbiboption{#1}\_iftrue\_def\_tmp{#2}\_else\_def\_tmp{#3}\_fi\_tmp}
\_def\_biboptionvalue#1#2{\_def\_tmp##1 #1:##2 ##3\_relax{\_def#2{##2}}%
   \_ea\_tmp\_biboptionsi #1: \_relax}

\_def\_readbiboptions{%
   \_RetrieveFieldIn{option}\_biboptionsi
   \_toks1=\_ea{\_biboptionsi}%
   \_edef\_biboptionsi{\_space \_the\_toks1 \_space \_the\_biboptions \_space}%
}
```

Formatting of Author/Editor lists.

bib-iso690.opm

```
\_def\_firstauthorformat{%
   \_upper{\_Lastname}\_bprintc\_Firstname{, *}\_bprintc\_Von{ *}\_bprintc\_Junior{, *}%
}
\_def\_otherauthorformat{%
   \_bprintc\_Firstname{* }\_bprintc\_Von{* }\_upper{\_Lastname}\_bprintc\_Junior{, *}%
}
\_def\_commonname{%
  \_ifnum\_NameCount=1
     \_firstauthorformat
  \_else
     \_ifnum0\_namecount=\_NameCount
        \_ifx\_maybeetal\_empty \_bibconjunctionand\_else , \_fi
     \_else , \_fi
     \_otherauthorformat
  \_fi
}
\_def\_authorname{%
   \_ifx\_authlist\_undefined \_edef\_authlist{\_Lastname,\_Firstname,\_Von,\_Junior}%
   \_else \_edef\_authlist{\_authlist;\_Lastname,\_Firstname,\_Von,\_Junior}\_fi
   \_ifnum\_NameCount>0\_namecount\_relax\_else \_commonname \_fi
   \_ifnum\_NameCount=0\_namecount\_relax \_maybeetal \_fi
}
\_def\_editorname{%
   \_ifnum\_NameCount>0\_namecount\_relax\_else \_commonname \_fi
   \_ifnum\_NameCount=0\_namecount\_relax \_maybeetal \_fi
}

\_def\_prepareauedoptions#1{%
```

```
   \_def\_maybeetal{}\_csname lb@abbreviatefalse\_endcsname
   \_biboptionvalue{#1max}\_authormax
   \_biboptionvalue{#1min}\_authormin
   \_biboptionvalue{#1pre}\_authorpre
   \_biboptionvalue{#1print}\_authorprint
   \_isbiboption{#1etal}\_iftrue \_def\_maybeetal{\_Mtext{bib.etal}}\_fi
   \_biboptionvalue{#1trim}\_autrim
   \_let\_namecountraw=\_namecount
   \_ifx\_authormax\_empty \_else
      \_ifnum 0\_authormax<0\_namecount
         \_edef\_namecount{\_ifx\_authormin\_empty\_authormax\_else\_authormin\_fi}%
         \_def\_maybeetal{\_Mtext{bib.etal}}%
   \_fi\_fi
   \_ifx\_autrim\_empty \_def\_autrim{10000}\_fi
   \_ifnum\_autrim=0 \_def\_autrim{10000}\_fi
   \_ifnum 0\_namecount<\_autrim\_relax \_else \_AbbreviateFirstname \_fi
}
\_def\_maybeetal{}

\_ifx\upper\_undefined
   \_ifx\caps \_undefined \_def\upper{\_uppercase\_ea}\_else
                          \_def\upper#1{{\caps\_rm #1}}\_fi
\_fi
\_let\_upper=\upper
```

Preparing bib-mark (used when \nonumcitations). 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 _createbibark macro in the \bibtexhook tokens list, if another bibmark format is needed. The macro _createbibmark must be expandable. See also OpTEX trick 0104.

bib-iso690.opm

```
\_def\_setbibmark{%
   \_ifx\_authlist\_undefined \_def\_authlist{,;}\_fi
   \_RetrieveFieldIn{bibmark}\_tmp
   \_ifx\_tmp\_empty
      \_RetrieveFieldIn{year}\_tmp
      \_edef\_tmp{\_ea\_createbibmark\_expanded{\_tmp;\_authlist};,;,;,;\_fin}\_fi
   \_bibmark=\_ea{\_tmp}%
}
\_def\_createbibmark #1;#2,#3;#4,#5\_fin{% #1=year #2=LastName #3=FirstName #4=nextAuthor
   #2\_ifx^#4^\_else \_Mtext{bib.etal}\_fi \_ifx^#1^\_else, #1\_fi
}
```

Setting phrases.

bib-iso690.opm

```
\_def\_bibconjunctionand{\_Mtext{bib.and}}
\_def\_preurl{\_Mtext{bib.available}}
\_let\_predoi=\_preurl
\_def\_postedition{\_mtext{bib.edition}}
\_def\_Inclause{In:~}
\_def\_prevolume{\_mtext{bib.volume}}
\_def\_prenumber{\_mtext{bib.number}}
\_def\_prepages{\_mtext{bib.prepages}}
\_def\_posteditor{\_ifnum0\_namecountraw>1 \_Mtext{bib.editors}\_else\_Mtext{bib.editor}\_fi}
```

_Mtext{⟨*identifier*⟩} expands to a phrase by outer document language (no entry language).

bib-iso690.opm

```
\_chardef\_documentlanguage=\_language
\_def\_Mtext#1{\_csname _mt:#1:\_csname _lan:\_the\_documentlanguage\_endcsname\_endcsname}

\_CreateField {lang}
\_def\_setlang#1{\_ifx#1\_empty \_else
     \_setbox0=\_vbox{\_langinput{#1}}%
     \_ifcsname _mt:bib.and:#1\_endcsname
        \_language=\_csname _#1Patt\_endcsname \_relax
     \_else \_opwarning{No phrases for "#1" used by [\EntryKey] in .bib}%
   \_fi\_fi
}
```

Sorting.

bib-iso690.opm

```
\_fieldalias {key} {sortedby}
```

Supporting macros.

bib-iso690.opm

```
\_def\_bibwarninga{\_bibwarning}
\_def\_bibwarningb{\_bibwarning}

\_def\_docitedate #1/#2/#3/#4\_relax{[\_Mtext{bib.citedate}%
   \_if^#2^#1\_else
      \_if^#3^#1/#2\_else
         \_cs{_\_cs{_lan:\_the\_documentlanguage}dateformat}#1/#2/#3\relax
   \_fi\_fi ]%
}
\_def\_doyear#1{
   \_biboptionvalue{yearprint}\_yearprint
   \_ifx\_yearprint\_empty#1\_else\_def\YEAR{#1}\_yearprint\_fi
}
\_def\_preparenumbering{%
   \_def\VOL{\_RetrieveField{volume}}%
   \_def\NO{\_RetrieveField{number}}%
   \_def\PP{\_RetrieveField{pages}}%
}
\_def\_prepareednote{%
   \_def\EDN{\_RetrieveField{edition}}%
   \_def\ADDR{\_RetrieveField{address}}%
   \_def\PUBL{\_RetrieveField{publisher}}%
   \_def\YEAR{\_RetrieveField{year}}%
   \_def\AU{\_bprintb[!author]{\_doauthor0{####1}}{}}%
   \_def\ED{\_bprintb[!editor]{\_doeditor0{####1}}{}}%
   \_preparenumbering
}
\_def\_doedition#1{%
   \_biboptionvalue{editionprint}\_editionprint
   \_ifx\_editionprint\_empty#1\_postedition\_else\_def\ED{#1}\_editionprint\_fi
}
\_def\_doauthor#1#2{\_prepareauedoptions{au}\_let\_iseditorlist=\_undefined
   \_if1#1\_def\AU{#2}\_else\_let\_authorprint=\_empty\_fi
   \_ifx\_authorprint\_empty #2\_else \_authorprint\_fi
}
\_def\_doeditor#1#2{\_prepareauedoptions{ed}\_let\_firstauthorformat=\_otherauthorformat
   \_if1#1\_def\ED{#2}\_else\_let\_authorprint=\_empty\_fi
   \_ifx\_authorprint\_empty #2\_posteditor\_else \_authorprint\_fi
}
```

Entry types.

bib-iso690.opm

```
\_sdef{_print:BEGIN}{%
   \_readbiboptions
   \_biboptionvalue{titlepost}\_titlepost
   \_isbiboption{unpublished}\_iftrue \_let\_bibwarninga=\_relax \_let\_bibwarningb=\_relax \_fi
   \_isbiboption{nowarn}\_iftrue \_let\_bibwarning=\_relax \_fi
   \_isbiboption{urlalso}\_iftrue \_def\_preurl{\_Mtext{bib.availablealso}}\_fi
   \_RetrieveFieldIn{lang}\_langentry \_setlang\_langentry
}
\_sdef{_print:END}{%
   \_bprinta [note]       {*.}{}%
   \_setbibmark
}
\_def\_bookgeneric#1{%
   \_bprinta [howpublished]  {[*].}{}%
   \_bprintb [edition]    {\_doedition{##1}}{}%
   \_bprinta [ednote]     {*.}{}%
   \_bprinta [address]    {*\_bprintv[publisher]{:}{\_bprintv[year]{,}{.}}}{\_bibwarninga}%
   \_bprinta [publisher]  {*\_bprintv[year]{,}{.}}{\_bibwarninga}%
   \_bprintb [year]       {\_doyear{##1}\_bprintv[citedate]{\_bprintv[numbering]{.}{}}{.}}%
                                                                                    {\_bibwarning}%
   \_bprinta [numbering]  {\_preparenumbering*\_bprintv[citedate]{}{}}{}%
   \_bprinta [citedate]   {\_docitedate*///\_relax.}{}%
   #1%
   \_bprinta [series]     {*.}{}%
   \_bprinta [isbn]       {ISBN~*.}{\_bibwarningb}%
   \_bprinta [issn]       {ISSN~*.}{}%
   \_bprintb [doi]        {\_predoi DOI \_ulink[http://dx.doi.org/##1]{##1}.}{}%
   \_bprintb [url]        {\_preurl\_url{##1}. }{}%
}
\_sdef{_print:book}{%
   \_bprintb [!author]    {\_doauthor1{##1}}{\_bibwarning}%
   \_bprintb [title]      {{\_em##1}\_bprintc\_titlepost{*}\_bprintv[howpublished]{}{}}%
                                                                                     {\_bibwarning}%
   \_bookgeneric{}%
}
\_sdef{_print:article}{%
   \_biboptionvalue{journalpost}\_journalpost
   \_bprintb [!author]   {\_doauthor1{##1}}{\_bibwarning}%
   \_bprinta [title]     {*.\_bprintc\_titlepost{*.}}{\_bibwarning}%
   \_bprintb [journal]   {{\_em##1}\_bprintc\_journalpost{*}\_bprintv[howpublished]{}{}}%
                                                                                     {\_bibwarninga}%
   \_bprinta [howpublished]  {[*].}{}%
   \_bprinta [address]   {*\_bprintb[publisher]{:}{,}}{}%
   \_bprinta [publisher] {*, }{}%
   \_bprinta [month]     {*, }{}%
   \_bprintb [year]      {\_doyear{##1}\_bprintv[volume,number,pages]{,}{}}{}%
   \_bprinta [numbering] {\_preparenumbering*\_bprintv[citedate]{}{}}
                        {\_bprinta [volume] {\_prevolume*\_bprintv[number,pages]{,}{}}{}%
                         \_bprinta [number] {\_prenumber*\_bprintv[pages]{,}{}}{}%
                         \_bprintb [pages]  {\_prepages\_hbox{##1}\_bprintv[citedate]{}{}}%
                                                                                     {\_bibwarninga}}%
   \_bprinta [citedate]  {\_docitedate*///\_relax.}{}%
   \_bprinta [issn]      {ISSN~*.}{}%
   \_bprintb [doi]       {\_predoi DOI \_ulink[http://dx.doi.org/##1]{##1}.}{}%
   \_bprintb [url]       {\_preurl\_url{##1}. }{}%
}
\_sdef{_print:inbook}{%
   \_let\_bibwarningb=\_relax
   \_bprintb [!author]   {\_doauthor1{##1}}{\_bibwarning}%
   \_bprinta [title]     {*.}{\_bibwarning}%
                        \_Inclause
   \_bprintb [!editor]   {\_doeditor1{##1}}{}%
   \_bprintb [booktitle] {{\_em##1}\_bprintc\_titlepost{*}\_bprintv[howpublished]{}{}}%
                                                                                     {\_bibwarning}%
   \_bookgeneric{\_bprintb [pages]  {\_prepages\_hbox{##1}. }{}}%
}
\_slet{_print:inproceedings}{_print:inbook}
\_slet{_print:conference}{_print:inbook}

\_sdef{_print:thesis}{%
   \_bprintb [!author]    {\_doauthor1{##1}}{\_bibwarning}%
   \_bprintb [title]      {{\_em##1}\_bprintc\_titlepost{*}\_bprintv[howpublished]{}{}}%
                                                                                     {\_bibwarning}%
   \_bprinta [howpublished]  {[*].}{}%
   \_bprinta [address]    {*\_bprintv[school]{:}{\_bprintv[year]{,}{.}}}{\_bibwarning}%
   \_bprinta [school]     {*\_bprintv[year]{,}{.}}{\_bibwarning}%
   \_bprinta [month]      {*, }{}%
   \_bprintb [year]       {\_doyear{##1}\_bprintv[citedate]{}{.}}{\_bibwarninga}%
   \_bprinta [citedate]   {\_docitedate*///\_relax.}{}%
   \_bprinta [type]       {*\_bprintv[ednote]{,}{.}}%
                         {\_ifx\_thesistype\_undefined\_bibwarning
                          \_else\_thesistype\_bprintv[ednote]{,}{.}\_fi}%
   \_bprinta [ednote]     {*.}{}%
   \_bprintb [doi]        {\_predoi DOI \_ulink[http://dx.doi.org/##1]{##1}.}{}%
   \_bprintb [url]        {\_preurl\_url{##1}. }{}%
}
\_sdef{_print:phdthesis}{\_def\_thesistype{\_Mtext{bib.phdthesis}}\_cs{_print:thesis}}
\_sdef{_print:mastersthesis}{\_def\_thesistype{\_Mtext{bib.masthesis}}\_cs{_print:thesis}}
\_sdef{_print:bachelorsthesis}{\_def\_thesistype{\_Mtext{bib.bachthesis}}\_cs{_print:thesis}}

\_sdef{_print:online}{%
```

```
   \_bprintb [!author]    {\_doauthor1{##1}}{}%
   \_bprintb [title]      {{\_em##1}\_bprintc\_titlepost{*}\_bprintv[howpublished]{}{}}%
                                                                                     {\_bibwarning}%
   \_bprinta [howpublished]  {[*].}{}%
   \_bprinta [ednote]     {\_prepareednote*\_bprintv[citedate]{}{.}}{}%
   \_bprinta [year]       {}{}%
   \_bprinta [accessed]   {\_docitedate*///\_relax.}{\_bibwarning}%
   \_bprintb [doi]        {\_predoi DOI \_ulink[http://dx.doi.org/##1]{##1}.}{}%
   \_bprintb [url]        {\_preurl\_url{##1}. }{\_bibwarning}%
}

\_sdef{_print:generic}{%
   \_bprintb [!author]    {\_doauthor1{##1}}{\_bibwarning}%
   \_bprintb [title]      {{\_em##1}\_bprintc\_titlepost{*}\_bprintv[howpublished]{}{}}%
                                                                                     {\_bibwarning}%
   \_bprinta [howpublished]  {[*].}{}%
   \_bprinta [ednote]     {\_prepareednote*\_bprintv[citedate]{}{.}}{\_bibwarning}%
   \_bprinta [citedate]   {\_docitedate*///\_relax.}{}%
   \_bprintb [doi]        {\_predoi DOI \_ulink[http://dx.doi.org/##1]{##1}.}{}%
   \_bprintb [url]        {\_preurl\_url{##1}. }{}%
}
\_slet{_print:booklet}{_print:generic}
\_slet{_print:incollection}{_print:generic}
\_slet{_print:manual}{_print:generic}
\_slet{_print:proceedings}{_print:generic}
\_slet{_print:techreport}{_print:generic}
\_slet{_print:unpublished}{_print:generic}

\_sdef{_print:misc}{\_let\_bibwarning=\_relax \_cs{_print:generic}}
```

2.33 Sorting and making Index

makeindex.opm

```
\_codedecl \makeindex {Makeindex and sorting <2024-12-01>} % preloaded in format
```

`\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 OpTeX 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.

makeindex.opm

```
\_def \_sortingdatalatin {%
   |,<,/,{ },_,-,&,@,%
   aAàÀâÂäÄáÁ,%
   ąĄ,%
   bB,%
   cC,%
   ćĆčČ,%
   dDďĎ,%
   eEèÈéÉëËêÊěĚ,%
   ęĘ,%
   fF,%
   gG,%
   hH,%
   ^^T^^U^^V,% ch Ch CH
   iIíÍïÏîÎ,%
   jJ,%
   kK,%
   lLĺĹľĽ,%
   łŁ,%
```

```
  mM,%
  nNňŇ,%
  ńŃñÑ,%
  oOöÖóÓôÔ,%
  pP,%
  qQ,%
  rRŕŔ,%
  řŘ,%
  sSß,%
  śŚšŠ,%
  tTťŤ,%
  uUùÙûÛüÜúÚůŮűŰ,%
  vV,%
  wW,%
  xX,%
  yYýÝÿŸ,%
  zZ,%
  žŽ,%
  źŹ,%
  żŻ,%
  ^^Z,% Hungarian: cz:c^^Z, etc., see \_compoundcharshu in lang-data.opm
  0,1,2,3,4,5,6,7,8,9,',>%
}
```

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.

makeindex.opm

```
\_def \_ignoredcharsgeneric  {.,;?!:'"()[]=+-}
```

Sorting is always processed by rules of a given language. The macros `\_sortingdata`⟨*lang-tag*⟩, `\_ignoredchars`⟨*lang-tag*⟩ and `\_compoundchars`⟨*lang-tag*⟩ declare these rules. The ⟨*lang-tag*⟩ is ISO code of the language: en, cs, de, pl, es for example. The English language is implemented here. Other languages are implemented in the `lang-data.opm` file (see section 2.37.4).

makeindex.opm

```
\_let \_sortingdataen = \_sortingdatalatin   % English alphabet is subset of Latin
\_let \_ignoredcharsen = \_ignoredcharsgeneric
\_def \_compoundcharsen {}  % English doesn't have compound characters like DZ
```

The `\_compoundchars`⟨*lang-tag*⟩ can declare changes performed before sorting. For example Czech language declares:

```
\_let \_sortingdatacs = \_sortingdatalatin  % Czech alphabet is subset of Latin
\_def \_compoundcharscs {ch:^^T Ch:^^U CH:^^V}
```

It transforms two-letters `ch` to single character `^^T` because ch is treated as single compound character by Czech rules and CH is sorted between H and I. See `\_sortingdatalatin` where `^^T` is used. This declaration makes more transformations of Ch and CH too. The declarations of the form `x:y` in the `\_compoundchars`⟨*lang-tag*⟩ are separated by space.

You can declare a transformation from single letter to more letters too. For example German rules sets ß equal to ss during sorting:

```
\_let \_sortingdatade = \_sortingdatalatin  % German alphabet is subset of Latin
\_def \_compoundcharsde {ß:ss}
```

If there are two words equal after first pass of sorting: Masse (mass) and Maße (measures) for example, then second pass must decide about the order. DIN 5007, section 6.1 says: ss must be before ß in this case. So, we want to switch off the `\_compoundchars` declaration for the second pass and use the order of s and ß given in `\_sortingdata`. This is possible if the `\_xcompoundchars`⟨*lang-tag*⟩ is defined. It has precedence in the second pass of sorting. We declare for German:

```
\_def \_xcompoundcharsde {}
```

Geman 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:

```
\_def \_compoundcharsde {ß:ss Ä:AE Ö:OE Ü:UE ä:ae ö:oe ü:ue}
```

Because u < ü in `\_sortingdata` and because `\_xcompoundcharsde` is empty, we have Mueller < Müller after second pass of the sorting.

You can declare these macros for more languages if you wish to use `\makeindex` with sorting rules with respect to your language. Note: if you need to map compound characters to a character, don't use `^^I`, `^^J` or `^^M` because these characters have very specific category codes.

If you created `\_sortingdata` etc. for your language, please, send them to me. I am ready to add them to the file `lang-data.opm` in a new OpTeX release. See also section 2.37.4.

French sorting rule says: if the words are the same except for accents then accented letters are sorted after unaccented leters but read the words from their end in the second pass. For example correct sorting is: cote < côte < coté < côté. This rule can be activated if the control sequence `\_secondpass`⟨*lang-tag*⟩ is set to `\_reversewords`. For example, `lang-data.opm` declares `\_let\_secondpassfr=\_reversewords`.

Preparing to primary pass is performed by the `\_setprimarysorting` macro implemented here. The ⟨*lang-tag*⟩ is saved to the `\_sortinglang` macro when sorting is initialized in `\_dosorting` (it is typically derived from current `\language` value). The `\_setprimarysorting` is called from `\_dosorting` macro and all processing of sorting is in a group. It sets actual `\_sortingdata`, `\_compoundchars` and `\_ignoredchars` if given language declares them. If not then warning will be printed using `\_nold` macro and English data are used. The `\lccode` of all characters from `\_sortingdata` and `\_ignoredchars` are set. The sorted words will be converted using `\_compoundchars` followed by `\lowercase` before first pass is run.

makeindex.opm

```
\_def\_setprimarysorting {%
   \_ea\_let \_ea\_sortingdata \_csname _sortingdata\_sortinglang\_endcsname
   \_ea\_let \_ea\_compoundchars \_csname _compoundchars\_sortinglang\_endcsname
   \_ea\_let \_ea\_ignoredchars \_csname _ignoredchars\_sortinglang\_endcsname
   \_def\_nold{}%
   \_ifx \_sortingdata\_relax \_addto\_nold{ sortingdata}%
       \_let \_sortingdata = \_sortingdataen \_fi
   \_ifx \_compoundchars\_relax \_addto\_nold{ compoundchars}%
       \_let \_compoundchars = \_compoundcharsen \_fi
   \_ifx \_ignoredchars\_relax \_addto\_nold{ ignoredchars}%
       \_let \_ignoredchars = \_ignoredcharsen \_fi
   \_ifx\_nold\_empty\_else \_opwarning{Missing\_nold\_space for language (\_sortinglang)}\_fi
   \_ifx \_compoundchars\_empty \_else
      \_edef \_compoundchars {\_detokenize\_ea{\_compoundchars} }\_fi % all must be catcode 12
   \_def \_act ##1{\_ifx##1\_relax \_else
      \_ifx##1,\_advance\_tmpnum by1
      \_else \_lccode`##1=\_tmpnum \_fi
      \_ea\_act \_fi}%
   \_tmpnum=65 \_ea\_act \_sortingdata \_relax
   \_def \_act ##1{\_ifx##1\_relax \_else
      \_lccode`##1=`^^I
      \_ea\_act \_fi}%
   \_ea\_act \_ignoredchars \_relax
}
```

Preparing to secondary pass is implemented by the `\_setsecondarysorting` macro.

makeindex.opm

```
\_def\_setsecondarysorting {%
   \_def \_act ##1{\_ifx##1\_relax \_else
      \_ifx##1,\_else \_advance\_tmpnum by1 \_lccode`##1=\_tmpnum \_fi
      \_ea\_act \_fi}%
  \_tmpnum=64 \_ea\_act \_sortingdata \_relax
}
```

Strings to be sorted are prepared in `\,`⟨*string*⟩ control sequences (to save `\TeX` memory). The `\_preparesorting` `\,`⟨*string*⟩ converts ⟨*string*⟩ to `\_tmpb` with respect to the data initialized in `\_setprimarysorting` or `\_setsecondarysorting`.
The part of the string after `^^^` is ignored (you can have the same sorting key for different things) and the compoud characters are converted by the `\_docompound` macro.

makeindex.opm

```
\_def \_preparesorting #1{%
   \_edef \_tmpb {\_ea\_ignoreit\_csstring #1}%        <string> -> <string>
   \_edef\_tmpb{\_ea \_stripfromcaret \_tmpb ^^^\_fin}% <string>^^^<ignore> -> <string>
   \_ea \_docompound \_compoundchars \_relax:{}        % replace compound characters
   \_lowercase \_ea{\_ea\_def \_ea\_tmpb \_ea{\_tmpb}}% convert in respect to \_sortingdata
   \_ea\_replstring \_ea\_tmpb \_ea{\_csstring^I}{}% remove ignored characters
}
\_def \_docompound #1:#2 {%
   \_ifx\_relax#1\_else \_replstring\_tmpb {#1}{#2}\_ea\_docompound \_fi
}
\_def\_stripfromcaret #1^^^#2\_fin{#1}
```

Macro `\_isAleB` `\,`⟨*string1*⟩ `\,`⟨*string2*⟩ returns the result of comparison of given two strings to `\_ifAleB` control sequence. Usage: `\_isAleB \,`⟨*string1*⟩ `\,`⟨*string2*⟩ `\_ifAleB ... \_else ... \_fi` The converted strings (in respect of the data prepared for first pass) must be saved as values of `\,`⟨*string1*⟩ and `\,`⟨*string2*⟩ macros. The reason is speed: we don't want to convert them repeatedly in each comparison. The macro `\_testAleB` ⟨*converted-string1*⟩`&\_relax`⟨*converted-string2*⟩`&\_relax \,`⟨*string1*⟩`\,`⟨*string2*⟩ does the real work. It reads the first character from both converted strings, compares them and if it is equal then calls itself recursively else gives the result.

makeindex.opm

```
\_newifi \_ifAleB

\_def\_isAleB #1#2{%
   \_edef\_tmpb {#1&\_relax#2&\_relax}%
   \_ea \_testAleB \_tmpb #1#2%
}
\_def\_testAleB #1#2\_relax #3#4\_relax #5#6{%
  \_if #1#3\_if #1&\_testAleBsecondary #5#6%   goto to the second pass::
          \_else \_testAleB #2\_relax #4\_relax #5#6%
          \_fi
  \_else \_ifnum `#1<`#3 \_AleBtrue \_else \_AleBfalse \_fi
  \_fi
}
```

The `\_testAleBsecondary` `\,`⟨*string1*⟩ `\,`⟨*string2*⟩ is run if the words are equal in the primary pass. It runs `\_setsecondarysorting` if it was not initialized already. Then prepares compared words to `\_tmpa` and `\_tmpb` and corrects them by `\_prepsecondpass` if needed. Finally, the test is recursively done by the macro `\_testAleBsecondaryX` ⟨*converted-string1*⟩`0\_relax`⟨*converted-string2*⟩`1\_relax`

makeindex.opm

```

\_def\_testAleBsecondary#1#2{%
  \_setsecondarysorting \_let\_setsecondarysorting=\_relax
  \_preparesorting#1\_let\_tmpa=\_tmpb \_preparesorting#2%
  \_prepsecondpass
  \_edef\_tmpb{\_tmpa0\_relax\_tmpb1\_relax}%
  \_ea\_testAleBsecondaryX \_tmpb
}
\_def\_testAleBsecondaryX #1#2\_relax #3#4\_relax {%
   \_if #1#3\_testAleBsecondaryX #2\_relax #4\_relax
   \_else \_ifnum `#1<`#3 \_AleBtrue \_else \_AleBfalse \_fi
   \_fi
}
```

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.

makeindex.opm

```
\_def\_mergesort #1#2,#3{% by Miroslav Olsak
   \_ifx,#1%                       % prazdna-skupina,neco,  (#2=neco #3=pokracovani)
      \_toksapp\_tmptoks{#2,}%        % dvojice skupin vyresena
      \_sortreturn{\_fif\_mergesort#3}%   % \mergesort pokracovani
   \_fi
   \_ifx,#3%                       % neco,prazna-skupina,  (#1#2=neco #3=,)
      \_toksapp\_tmptoks{#1#2,}%      % dvojice skupin vyresena
      \_sortreturn{\_fif\_mergesort}% % \mergesort dalsi
   \_fi
   \_ifx\_fin#3%                    % neco,konec (#1#2=neco)
      \_if;\_the\_tmptoks;%           % neco=kompletni setrideny seznam
```

```
         \_tmptoks{#1#2}%
         \_sortreturn{\_fif\_fif\_gobbletoend}%   % koncim
      \_else                      % neco=posledni skupina nebo \end
         \_sortreturn{\_fif\_fif       % spojim \indexbuffer+necoa cele znova
                      \_tmptoks\_ea{\_ea}\_ea\_mergesort\_the\_tmptoks#1#2,#3}%
   \_fi\_fi                      % zatriduji: p1+neco1,p2+neco2, (#1#2=p1+neco1 #3=p2)
   \_isAleB #1#3\_ifAleB        % p1<p2
      \_toksapp\_tmptoks{#1}%    % p1 do bufferu
      \_sortreturn{\_fif\_mergesort#2,#3}%         % \mergesort neco1,p2+neco2,
   \_else                       % p1>p2
      \_toksapp\_tmptoks{#3}%   % p2 do bufferu
      \_sortreturn{\_fif\_mergesort#1#2,}%         % \mergesort p1+neco1,neco2,
   \_fi
   \_relax % zarazka, na ktere se zastavi \sortreturn
}
\_def\_sortreturn#1#2\_fi\_relax{#1} \_def\_fif{\_fi}
\_def\_gobbletoend #1\_fin{}
```

The _dosorting `\list` macro redefines `\list` as sorted `\list`. The `\list` have to include control sequences in the form `\`⟨*c*⟩⟨*string*⟩. These control sequences will be sorted with respect to ⟨*strings*⟩ without change of meanings of these control sequences. Their meanings are irrelevant when sorting. The first character ⟨*c*⟩ in `\`⟨*c*⟩⟨*string*⟩ should be whatever. It does not influence the sorting. OpTeX uses comma at this place for sorting indexes: `\,`⟨*word1*⟩ `\,`⟨*word2*⟩ `\,`⟨*word3*⟩ `...`.

The current language (chosen for hyphenation patterns) is used for sorting data. If the macro _sortinglang is defined as ⟨*lang-tag*⟩ (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.

makeindex.opm

```
\_newifi \_ifasciisorting  \_asciisortingfalse
\_def\_dosorting #1{%
   \_begingroup
      \_ifasciisorting \_def\_sortinglang{ASCII}\_fi
      \_ifx\_sortinglang\_undefined \_edef\_sortinglang{\_cs{_lan:\_the\_language}}\_fi
      \_sortmessage{OpTeX: Sorting \_string#1 (\_sortinglang) ...^^J}%
      \_ismacro\_sortinglang{ASCII}\_iftrue
          \_def \_preparesorting##1{\_edef\_tmpb{\_ea\_ignoreit\_csstring##1}}%
          \_let \_setsecondarysorting=\_relax
      \_else
         \_setprimarysorting
      \_fi
      \_def \_act##1{\_preparesorting ##1\_edef##1{\_tmpb}}%
      \_ea\_xargs \_ea\_act #1;% \_preparesorting for first pass of sorting applied
      \_ifcsname _xcompoundchars\_sortinglang\_endcsname
          \_ea\_let \_ea\_compoundchars \_csname _xcompoundchars\_sortinglang\_endcsname
      \_fi  % \_compoundchars can differ in the second pass of sorting
      \_csname _secondpass\_sortinglang \_endcsname % activates \_reversewords if needed
      \_def \_act##1{\_toksapp\_tmptoks{##1,}}%
      \_tmptoks{}%
      \_edef #1{\_ea}\_ea\_xargs \_ea\_act #1;% commas between items added, mergesort initialized
      \_tmptoks\_ea{\_ea}\_ea\_mergesort \_the\_tmptoks\_fin,\_fin
   \_ea\_endgroup
   \_ea\_def\_ea#1\_ea{\_the\_tmptoks}%
}
\_let\_sortmessage=\_message % User can do \let\_sortmessage=\_ignoreit, for example
```

French rules needs reverse reading the words in the second pass. The _reversewords is activated in this case and it adds new job to the macro _prepsecondpass: it reverses the letters in the compared words (saved in `\_tmpa` and `\_tmpb`) by the expandable _sortrevers macro. The _prepsecondpass macro is used in the _testAleBsecondary and it is empty by default.

makeindex.opm

```
\_def\_prepsecondpass{}
\_def\_reversewords{%
   \_addto\_prepsecondpass{\_edef\_tmpa{\_ea\_sortrevers\_tmpa\_relax}%
                           \_edef\_tmpb{\_ea\_sortrevers\_tmpb\_relax}}%
}
\_def\_sortrevers #1#2\_relax{\_ifx^#2^#1\_else \_sortrevers#2\_relax #1\_fi}
```

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`.

makeindex.opm

```
\_def\_makeindex{\_par
  \_ifx\_iilist\_empty \_opwarning{index data-buffer is empty. TeX me again}%
  \_incr\_unresolvedrefs
  \_else
    \_dosorting \_iilist % sorting \_iilist
    \_bgroup
       \_rightskip=0pt plus1fil \_exhyphenpenalty=10000 \_leftskip=\_iindent
       \_ea\_xargs \_ea\_printindexitem \_iilist ;\_par
    \_egroup
  \_fi
}
\_public \makeindex ;
```

The `\_printindexitem` `\,`⟨*word*⟩ prints one item to the index. If `\_,`⟨*word*⟩ is defined then this is used instead real ⟨*word*⟩ (this exception is declared by `\iis` macro). Else ⟨*word*⟩ is printed by `\_printii`. Finally, `\_printiipages` prints the value of `\,`⟨*word*⟩, i.e. the list of pages.

makeindex.opm

```
\_def\_printindexitem #1{%
   \_ifcsname _\_csstring #1\_endcsname
      \_ea\_ea\_ea \_printii \_csname _\_csstring #1\_endcsname &%
   \_else
      \_ea\_ea\_ea\_printii \_ea\_ignoreit \_csstring #1&%
   \_fi
   \_ea\_printiipages #1&
}
```

`\_printii` ⟨*word*⟩`&` does more intelligent work because we are working with words in the form ⟨*main-word*⟩`/`⟨*sub-word*⟩`/`⟨*sub-sub-word*⟩. The `\everyii` tokens register is applied before `\noindent`. User can declare something special here.

The `\_newiiletter{`⟨*letter*⟩`}{`⟨*word*⟩`}` macro is empty by default. It is invoked if first letter of index entry is changed. You can declare a design between index entries here. You can try, for example:

```
\def\_newiiletter#1#2{%
    \bigskip \hbox{\setfontsize{at15pt}\bf #1}\nobreak\medskip}
```

`\_definefirstii` ⟨*word*⟩`&` macro defines `\_firstii` which is used as the ⟨*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{\bgroup \iicodes \uppercase{\egroup\def\_firstii{#1}}}}
\def\iicodes{}
\def\setiicodes #1#2,{\_ifx^#1^\_else
   \foreach #2\do{\_addto\iicodes{\uccode`##1=`#1}}
   \_ea\setiicodes \_fi
}
\setiicodes AÀÂÄÁ,ĆČ,DĎ,EÈÉËÊĚ,IÍÏÎ,LĹĽ,OÖÓÔ,RŔ,ŚŠ,TŤ,UÙÛÜÚŮŰ,YÝŸ,{},
```

If the first character of the ⟨*word*⟩ is < or > then it isn't printed. It can be used as first character of the index entry in order to put the entry to a group entries sorted <before or >after whole normal entries.

makeindex.opm

```
\_def\_printii #1&{\_definefirstii #1&%
   \_ifx\_firstii\_lastii\_else
      \_ea\_newiiletter\_ea{\_firstii}{#1}\_let\_lastii=\_firstii\_fi
   \_gdef\_currii{#1}\_the\_everyii\_noindent
   \_hskip-\_iindent \_ignorespaces \_isnextchar<{\_ea\_printiiA\_ignoreit}%
      {\_isnextchar>{\_ea\_printiiA\_ignoreit}{\_printiiA}}#1//}
\_def\_printiiA #1/{\_if^#1^\_let\_previi=\_currii \_else
   \_ea\_scanprevii\_previi/&\_edef\_tmpb{\_detokenize{#1}}%
   \_ifx\_tmpa\_tmpb \_iiemdash \_else#1 \_gdef\_previi{}\_fi
   \_ea\_printiiA\_fi
}
\_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}}}
\_def\_previi{} % previous index item
```

`\_printiipages` ⟨*pglist*⟩`&` gets ⟨*pglist*⟩ in the form ⟨*pg*⟩`:`⟨*type*⟩`,`⟨*pg*⟩`:`⟨*type*⟩`,...`⟨*pg*⟩`:`⟨*type*⟩ and it converts them to ⟨*pg*⟩`, `⟨*pg*⟩`, `⟨*from*⟩`--`⟨*to*⟩`, `⟨*pg*⟩ etc. The same pages must be printed only once and continuous consequences of pages must be compressed to the form ⟨*from*⟩-⟨*to*⟩. Moreover, the consequence is continuous only if all pages have the same ⟨*type*⟩. Empty ⟨*type*⟩ is most common, pages with `b` ⟨*type*⟩ must be printed as bold and with `i` ⟨*type*⟩ as italics. Moreover, the ⟨*pg*⟩ mentioned here are ⟨*gpageno*⟩, but we have to print ⟨*pageno*⟩. The following macros solve these tasks.

makeindex.opm

```
\_def\_printiipages#1&{\_let\_pgtype=\_undefined \_tmpnum=0 \_printpages #1,:,\_par}
\_def\_printpages#1:#2,{%  state automaton for compriming pages
   \_ifx,#1,\_uselastpgnum
   \_else \_def\_tmpa{#2}%
      \_ifx\_pgtype\_tmpa \_else
         \_let\_pgtype=\_tmpa
         \_uselastpgnum \_usepgcomma \_pgprint#1:{#2}%
         \_tmpnum=#1 \_returnfi \_fi
      \_ifnum\_tmpnum=#1 \_returnfi \_fi
      \_advance\_tmpnum by1
      \_ifnum\_tmpnum=#1 \_ifx\_lastpgnum\_undefined \_usepgdash\_fi
                        \_edef\_lastpgnum{\_the\_tmpnum:{\_pgtype}}%
                        \_returnfi \_fi
      \_uselastpgnum \_usepgcomma \_pgprint#1:{#2}%
      \_tmpnum=#1
      \_relax
   \_ea\_printpages \_fi
}
\_def\_returnfi #1\_relax{\_fi}
\_def\_uselastpgnum{\_ifx\_lastpgnum\_undefined
   \_else \_ea\_pgprint\_lastpgnum \_let\_lastpgnum=\_undefined \_fi
}
\_def\_usepgcomma{\_ifnum\_tmpnum>0, \_fi} % comma+space between page numbers
\_def\_usepgdash{\_hbox{--}}               % dash in the <from>--<to> form
```

You can re-define `\_pgprint` ⟨*gpageno*⟩`:{`⟨*iitype*⟩`}` if you need to implement more ⟨*iitypes*⟩.

makeindex.opm

```
\_def\_pgprint #1:#2{%
   \_ifx ,#2,\_pgprintA{#1}\_returnfi \_fi
   \_ifx b#2{\_bf \_pgprintA{#1}}\_returnfi \_fi
   \_ifx i#2{\_it \_pgprintA{#1}}\_returnfi \_fi
   \_ifx u#2\_pgu{\_pgprintA{#1}}\_returnfi \_fi
  \_pgprintA{#1}\_relax
}
\_def\_pgprintA #1{\_ilink[pg:#1]{\_cs{_pgi:#1}}} % \ilink[pg:<gpageno>]{<pageno>}
\_def\_pgu#1{\_leavevmode\_vtop{\_hbox{#1}\kern.3ex\_hrule}}
```

The `\iindex{`⟨*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 `\iindex`.

makeindex.opm

```
\_def\_iindex#1{\_isempty{#1}\_iffalse
   \_openref{\_def~{ }\_ewref\_Xindex{{#1}{\_iitypesaved}}}\_fi}
\_public \iindex ;
```

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*⟩.

makeindex.opm

```
\_def \_iilist {}
\_def \_Xindex #1#2{\_ea\_XindexA \_csname ,#1\_ea\_endcsname \_currpage {#2}}
\_def \_XindexA #1#2#3#4{% #1=<word> #2=<gpageno> #3=<pageno> #4=<iitype>
   \_ifx#1\_relax \_global\_addto \_iilist {#1}%
                 \_gdef #1{#2:#4}%
   \_else \_global\_addto #1{,#2:#4}%
   \_fi
   \_sxdef{_pgi:#2}{#3}%
}
```

The implementation of macros `\ii`, `\iid`, `\iis` follows. Note that `\ii` works in the horizontal mode in order to the `\write` whatsit is not broken from the following word. If you need to keep vertical mode, use `\iindex`{⟨*word*⟩} directly.

The `\iitype` {⟨*type*⟩} saves the ⟨*type*⟩ to the `\_iitypesaved` macro. It is used in the `\iindex` macro.

makeindex.opm

```
\_def\_ii #1 {\_leavevmode\_def\_tmp{#1}\_iiA #1,,\_def\_iitypesaved{}}

\_def\_iiA #1,{\_if$#1$\_else\_def\_tmpa{#1}%
   \_ifx\_tmpa\_iiatsign \_ea\_iiB\_tmp,,\_else\_iindex{#1}\_fi
   \_ea\_iiA\_fi}
\_def\_iiatsign{@}

\_def\_iiB #1,{\_if$#1$\_else \_iiC#1/\_relax \_ea\_iiB\_fi}
\_def\_iiC #1/#2\_relax{\_if$#2$\_else\_iindex{#2#1}\_fi}

\_def\_iid #1 {\_leavevmode\_iindex{#1}\_def\_iitypesaved{}%
   \_isnextchar<{\_ignoreit}{\_isnextchar>{\_ignoreit}{}}#1\_futurelet\_tmp\_iiD}
\_def\_iiD{\_ifx\_tmp,\_else\_ifx\_tmp.\_else\_space\_fi\_fi}

\_def\_iis #1 #2{{\_def~{ }\_global\_sdef{_,#1}{#2}}\_ignorespaces}

\_def\_iitypesaved{}
\_def\_iitype #1{\_def\_iitypesaved{#1}\_ignorespaces}

\_public \ii \iid \iis \iitype ;
```

2.34 Footnotes and marginal notes

fnotes.opm

```
\_codedecl \fnote {Footnotes, marginal notes OpTeX <2023-04-15>} % preloaded in format
```

`\_gfnotenum` is a counter which counts footnotes globally in the whole document.

`\_lfnotenum` is a counter which counts footnotes at each chapter from one. It is used for local page footnote counters too.

`\_ifpgfnote` says that footnote numbers are counted on each page from one. We need to run `\openref` in this case.

`\fnotenum` is a macro that expands to footnote number counted in declared part.

`\fnotenumchapters` declares footnotes numbered in each chapter from one (default), `\fnotenumglobal` declares footnotes numbered in whole document from one and `\fnotenumpages` declares footnotes numbered at each page from one.

fnotes.opm

```
\_newcount\_gfnotenum \_gfnotenum=0
\_newcount\_lfnotenum

\_newifi \_ifpgfnote
\_def \_fnotenumglobal   {\_def\_fnotenum{\_the\_gfnotenum}\_pgfnotefalse}
\_def \_fnotenumchapters {\_def\_fnotenum{\_the\_lfnotenum}\_pgfnotefalse}
\_def \_fnotenumpages    {\_def\_fnotenum{\_trycs{_fn:\_the\_gfnotenum}{?}}\_pgfnotetrue}
\_fnotenumchapters  % default are footnotes counted from one in each chapter
\_def \fnotenum{\_fnotenum}
\_public \fnotenumglobal \fnotenumchapters \fnotenumpages ;
\_let \runningfnotes = \fnotenumglobal % for backward compatibility
```

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).

`fnotes.opm`

```
\_def \_printfnotemark  {\_quitvmode\_hbox{$^{\_fnotenum}$}}   % default footnote mark
\_def \_printfnotemarkA {\_printfnotemark}  % footnote marks used in text
\_def \_printfnotemarkB {\_printfnotemark}  % footnote marks used in front of footnotes

\_def \_fnotelinks#1#2{% <inText color> <inFootnote color>
   \_def\_printfnotemarkA{\_link[fnt:\_the\_gfnotenum]{#1}{\_printfnotemark}%
                         \_dest[fnf:\_the\_gfnotenum]}%
   \_def\_printfnotemarkB{\_link[fnf:\_the\_gfnotenum]{#2}{\_printfnotemark}%
                         \_dest[fnt:\_the\_gfnotenum]}%
}
\public \fnotelinks ;
```

Each footnote saves the `\_Xfnote` (without parameter) to the `.ref` file (if `\openref`). We can create the mapping from ⟨*gfnotenum*⟩ to⟨*pgfnotenum*⟩ in the macro `\_fn:`⟨*fnotenum*⟩. Each `\_Xpage` macro sets the `\_lfnotenum` to zero.

`fnotes.opm`

```
\_def \_Xfnote {\_incr\_lfnotenum \_incr\_gfnotenum
   \_sxdef{_fn:\_the\_gfnotenum}{\_the\_lfnotenum}}
```

The `\fnote` `{`⟨*text*⟩`}` macro is simple, `\fnotemark` and `\fnotetext` does the real work.

`fnotes.opm`

```
\_def\_fnote{\_fnotemark1\_fnotetext}
\_def\_fnotemark#1{{\_advance\_gfnotenum by#1\_advance\_lfnotenum by#1\_relax \_printfnotemarkA}}
```

The `\fnotetext` calls `\_opfootnote` which is equivalent to plain TeX `\vfootnote`. It creates new data to Insert `\footins`. The only difference is that we can propagate a macro parameter into the Insert group before the text is printed (see section 2.18). This propagated macro is `\_fnset` which sets smaller fonts.

Note that `\vfootnote` and `\_opfootnote` don't read the text as a parameter but during the normal horizontal mode. This is the reason why catcode changes (for example in-line verbatim) can be used here.

`fnotes.opm`

```
\_def\_fnotetext{\_incr\_gfnotenum \_incr\_lfnotenum % global increment
   \_ifpgfnote \_openref \_fi
   \_wref \_Xfnote{}%
   \_ifpgfnote \_ifcsname _fn:\_the\_gfnotenum \_endcsname \_else
       \_opwarning{unknown \_noexpand\fnote mark. TeX me again}%
       \_incr\_unresolvedrefs
   \_fi\_fi
   \_opfootnote\_fnset\_printfnotemarkB
}
\_def\_fnset{\_everypar={}\_scalemain \_typoscale[800/800]}

\_public \fnote \fnotemark \fnotetext ;
```

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`$\le$`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.

`fnotes.opm`

```
\_newcount\_mnotenum    \_mnotenum=0        % global counter of mnotes
\_def \_Xmnote {\_incr\_mnotenum \_ea \_XmnoteA \_currpage}
\_def \_XmnoteA #1#2{% #1=<gpageno> #2=<pageno>
   \_sxdef{_mn:\_the\_mnotenum}{\_ifodd\_numtype{#2}{#1} \_right \_else \_left \_fi}}
\_def \_numtype #1#2{\_ifnum 0<0#1 #1\_else #2\_fi}
```

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.

fnotes.opm

```
\_def \_fixmnotes #1{\_edef\_mnotesfixed{\_cs{_\_csstring #1}}}
\_public \fixmnotes ;
```

The `\_mnoteD{`⟨*text*⟩`}` macro sets the position of the marginal note. The outer box of marginal note has zero width and zero depth and it is appended after current line using `\vadjust` primitive or it is inverted to vertical mode as a box shifted down by `\parskip` and with `\vskip-\baselineskip` followed.

fnotes.opm

```
\_def\_mnote #1#{\_ifx^#1^\_else \_mnoteC#1\_fin \_fi \_mnoteD}
\_def\_mnoteC up#1\_fin{\_mnoteskip=#1\_relax} % \mnote up<dimen> {<text>} syntax
\_long\_def\_mnoteD#1{%
   \_ifvmode \_vskip\_parskip{\_mnoteA{#1}}\_nobreak\_vskip-\_baselineskip\_vskip-\_parskip \_else
   \_lower\_dp\_strutbox\_hbox{}\_vadjust{\_kern-\_dp\_strutbox \_mnoteA{#1}\_kern\_dp\_strutbox}%
   \_fi
}
\_public \mnote ;
```

The `\mnoteskip` is a dimen value that denotes the vertical shift of marginal note from its normal position. A positive value means shift up, negative down. The `\mnoteskip` register is set to zero after the marginal note is printed. The new syntax `\mnote up`⟨*dimen*⟩`{`⟨*text*⟩`}` is possible too, but public `\mnoteskip` is kept for backward compatibility.

fnotes.opm

```
\_newdimen\_mnoteskip
\_public \mnoteskip ;
```

The `\_mnoteA` macro does the real work. The `\_lrmnote{`⟨*left*⟩`}{`⟨*right*⟩`}` uses only first or only second parameter depending on the left or right marginal note.

fnotes.opm

```
\_long\_def\_mnoteA #1{\_incr\_mnotenum
   \_ifx\_mnotesfixed\_undefined
      \_ifcsname _mn:\_the\_mnotenum \_endcsname
          \_edef\_mnotesfixed{\_cs{_mn:\_the\_mnotenum}}%
      \_else
          \_opwarning{unknown \_noexpand\mnote side. TeX me again}\_openref
          \_incr\_unresolvedrefs
          \_def\_mnotesfixed{\_right}%
   \_fi\_fi
   \_hbox to0pt{\_wref\_Xmnote{}\_everypar={}%
      \_lrmnote{\_kern-\_mnotesize \_kern-\_mnoteindent}{\_kern\_hsize \_kern\_mnoteindent}%
      \_vbox to0pt{\_vss \_setbox0=\_vtop{\_hsize=\_mnotesize
             \_lrmnote{\_leftskip=0pt plus 1fill \_rightskip=0pt}
                       {\_rightskip=0pt plus 1fil \_leftskip=0pt}%
             {\_the\_everymnote\_noindent#1\_endgraf}}%
          \_dp0=0pt \_box0 \_kern\_mnoteskip \_global\_mnoteskip=0pt}\_hss}%
}
\_def \_lrmnote#1#2{\_ea\_ifx\_mnotesfixed\_left #1\_else #2\_fi}
```

We don't want to process `\fnote`, `\fnotemark`, `\mnote` in TOC, headlines nor outlines.

fnotes.opm

```
\_regmacro {\_def\fnote#1{}} {\_def\fnote#1{}} {\_def\fnote#1{}}
\_regmacro {\_def\fnotemark#1{}} {\_def\fnotemark#1{}} {\_def\fnotemark#1{}}
\_regmacro {\_def\mnote#1{}} {\_def\mnote#1{}} {\_def\mnote#1{}}
```

2.35 Styles

OpTEX provides three styles: `\report`, `\letter` and `\slides`. Their behavior is documented in user part of the manual in the section 1.7.2 and `\slides` style (for presentations) is documented in `op-slides.pdf` which is an example of the presentation.

2.35.1 \report and \letter styles

styles.opm

```
\_codedecl \report {Basic styles of OpTeX <2021-03-10>} % preloaded in format
```

We define auxiliary macro first (used by the `\address` macro)
The `{\boxlines` ⟨*line-1*⟩⟨*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 `}`.

styles.opm

```
\_def\_boxlines{%
   \_def\_boxlinesE{\_ifhmode\_egroup\_empty\_fi}%
   \_def\_nl{\_boxlinesE}%
   \_bgroup \_lccode`=`^M\_lowercase{\_egroup\_let~}\_boxlinesE
   \_everypar{\_setbox0=\_lastbox\_endgraf
      \_hbox\_bgroup \_catcode`^M=13 \_let\_par=\_nl \_aftergroup\_boxlinesC}%
}
\_def\_boxlinesC{\_futurelet\_next\_boxlinesD}
\_def\_boxlinesD{\_ifx\_next\_empty\_else\_ea\_egroup\_fi}

\_public \boxlines ;
```

The `\report` style initialization macro is defined here.

styles.opm

```
\_def\_report{
   \_typosize[11/13.2]
   \_vsize=\_dimexpr \_topskip + 52\_baselineskip \_relax % added 2020-03-28
   \_let\_titfont=\_chapfont
   \_titskip=3ex
   \_eoldef\_author##1{\_removelastskip\_bigskip
      {\_leftskip=0pt plus1fill \_rightskip=\_leftskip \_it \_noindent ##1\_par}\_nobreak\_bigskip
   }
   \_public \author ;
   \_parindent=1.2em \_iindent=\_parindent \_ttindent=\_parindent
   \_footline={\_global\_footline={\_hss\_rmfixed\_folio\_hss}}
}
```

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.

styles.opm

```
\_def\_letter{
   \_def\_address{\_vtop\_bgroup\_boxlines \_parskip=0pt \_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.

styles.opm

```
\_def\_slides{\_par
  \_opinput{slides.opm}
  \_adef*{\_relax\_ifmmode*\_else\_ea\_startitem\_fi}
}
\_public \slides ;
```

2.35.2 \slides style for presentations

slides.opm

```
\_codedecl \slideshow {Slides style for OpTeX <2025-02-27>} % loaded on demand by \slides
```

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.

slides.opm

```
\_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.

slides.opm

```
\_footlinedist=1mm
\_footline={\_hss \_rlap{%
   \_rlap{\Grey\_kern.2\_hoffset\_vrule height6mm depth2mm width.8\_hoffset}%
                      \_hbox to\_hoffset{\White\_hss\_folio\_kern3mm}}}
```

The `\subtit` is defined analogically like `\tit`.

slides.opm

```
\_eoldef\_subtit#1{\_vskip20pt {\_leftskip=0pt plus1fill \_rightskip=\_leftskip
   \_subtitfont #1\_nbpar}}
```

The `\pshow`⟨*num*⟩ prints the text in invisible (transparent) font when `\layernum<`⟨*num*⟩. For transparency we need to define special graphics states.

slides.opm

```
\_def\_Transparent {\_transparency255 }
\_public \Transparent ;

\_def\_use#1#2{\_ifnum\_layernum#1\_relax#2\_fi}
\_def\_pshow#1{\_use{=#1}\Red \_use{<#1}\_Transparent \_ignorespaces}
```

The main level list of items is activated here. The `\_item:X` and `\_item:x` are used and are re-defined here. If we are in a nested level of items and `\pg+` is used then `\egroups` macro expands to the right number of `\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+`.

slides.opm

```
\_newcount\_gilevel
\_def{*}
\_adef*{\_relax\_ifmmode*\_else\_ea\_startitem\_fi} % defined also in styles.opm
\_sdef{_item:X}{\_scolor\_raise.2ex\_fullrectangle{.8ex}\_kern.5em}
\_sdef{_item:x}{\_scolor\_raise.3ex\_fullrectangle{.6ex}\_kern.4em}
\_style X
\_def\_egroups{\_par\_global\_gilevel=\_ilevel \_egroup}
\_everylist={\_novspaces \_ifcase\_ilevel \_or \_style x \_else \_style - \_fi
   \_addto\_egroups{\_egroup}}
```

The default values of `\pg`, i. e. `\pg;`, `\pg+` and `\pg.` are very simple. They are used when `\slideshow` is not specified.

slides.opm

```
\_def\_pg#1{\_cs{_spg:#1}}
\_sdef{_spg:;}{\_vfil\_break \_lfnotenumreset}
\_sdef{_spg:.}{\_endslides}
\_sdef{_spg:+}{\_par}
```

The `\_endslides` is defined as `\_end` primitive (preceded 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.`.

slides.opm

```
\_def\_endslides{\_vfill \_supereject \_byehook \_end}
\_def\bye{\_pg.}
```

We need no numbers and no table of contents when using slides. The `\_printsec` macro is redefined in order the title is centered and typeset in `\_scolor`.

slides.opm

```
\_def\_titfont{\_typosize[42/60]\_boldify \_scolor}
\_def\_subtitfont{\_typosize[20/30]\_boldify}
\_def\_secfont{\_typosize[25/30]\_boldify \_scolor}

\_nonum \_notoc \_let\_resetnonumnotoc=\_relax
\_def\_printsec#1{\_par
   \_abovetitle{\_penalty-400}\_bigskip
   {\_secfont \_noindent \_leftskip=0pt plus1fill \_rightskip=\_leftskip
     \_printrefnum[@\_quad]#1\_nbpar}\_insertmark{#1}%
   \_nobreak \_belowtitle{\_medskip}%
}
```

When `\slideshow` is active then each page is opened by `\setbox\_slidepage=\vbox\bgroup` (roughly speaking) and closed by `\egroup`. The material is `\unvbox`ed and saved for the usage in the next usage if `\pg+` is in process. The `\_slidelayer` is incremented instead `\pageno` if `\pg+`. This counter is equal to `\count1`, so it is printed to the terminal and log file next to `\pageno`.

The code is somewhat more complicated when `\layers` is used. Then ⟨*layered-text*⟩ is saved to the `\_layertext` macro, the material before it is in `\_slidepage` box and the material after it is in `\_slidepageB` box. The pages are completed in the `\loop` which increments the `\layernum` register and prints page by the `\_printlayers`

slides.opm

```
\_newbox\_slidepage  \_newbox\_slidepageB
\_countdef\_slidelayer=1

\_def\_slideshow{\_slidelayer=1 \_slideshowactive
   \_let\slideopen=\_relax  % first wins
   \_setbox\_slidepage=\_vbox\_bgroup\_bgroup}

\_def\_slideshowactive{%
   \_sdef{_spg:;}{\_closepage \_global\_slidelayer=1 \_resetpage \_openslide}
   \_sdef{_spg:.}{\_closepage \_endslides}
   \_sdef{_spg:+}{\_closepage \_incr\_slidelayer \_decr\_pageno \_openslide}
   \_let\_layers=\_layersactive
   \_slidelinks % to prevent hyperlink-dests duplication
}
\_def\_openslide{\_setbox\_slidepage=\_vbox\_bgroup\_bgroup \_setilevel
   \_ifvoid\_slidepage \_else \_unvbox\_slidepage \_nointerlineskip\_lastbox \_fi}
\_def\_setilevel{\_loop \_decr\_gilevel \_ifnum\_gilevel<0 \_else \_begitems \_repeat}

\_def\_closepage{\_egroups \_egroup
   \_ifnum \_maxlayers=0 \_unvcopy\_slidepage \_vfil\_break
   \_else \_begingroup \_setwarnslides \_layernum=0
      \_loop
         \_ifnum\_layernum<\_maxlayers \_advance\_layernum by1
            \_printlayers \_vfil\_break
            \_ifnum\_layernum<\_maxlayers \_incr\_slidelayer \_decr\_pageno \_fi
      \_repeat
      \_global\_maxlayers=0
      \_incr\_layernum \_global\_setbox\_slidepage=\_vbox{\_printlayers}%
      \_endgroup
   \_fi}
\_def\_resetpage{%
   \_global\_setbox\_slidepage=\_box\_voidbox \_global\_setbox\_slidepageB=\_box\_voidbox
   \_lfnotenumreset
}
\_def\_setwarnslides{%
   \_def\pg##1{\_opwarning{\_string\pg##1 \_layersenv}\_def\pg####1{}}%
   \_def\layers##1 {\_opwarning{\_string\layers\_space \_layersenv}\_def\layers####1{}}%
}
\_def\_layersenv{cannot be inside \_string\layers...\_string\endlayers, ignored}

\_def\_printlayers{\_unvcopy\_slidepage \_prevdepth=\_dp\_slidepage
```

```
   {\_layertext \_endgraf}%
   \_vskip\_parskip
   \_unvcopy\_slidepageB
}
\_let\_destboxori=\_destbox

\_newcount\_layernum \_newcount\_maxlayers
\_maxlayers=0

\_long\_def\_layersactive #1 #2\endlayers{%
   \_par\_penalty0\_egroup\_egroup
   \_gdef\_layertext{\_settinglayer#2}%
   \_global\_maxlayers=#1
   \_setbox\_slidepageB=\_vbox\_bgroup\_bgroup
      \_setbox0=\_vbox{{\_layernum=1 \_globaldefs=-1 \_layertext\_endgraf}}\_prevdepth=\_dp0
}
\_public \subtit \slideshow \pg \wideformat \use \pshow \layernum ;
```

`\slideopen` should be used instead `\slideshow` to deactivate it but keep the borders of groups.

slides.opm

```
\_def\_slideopen{\_let\slideshow=\_relax % first wins
   \_sdef{_spg:;}{\_egroups\_vfil\_break \_lfnotenumreset\_bgroup \_setilevel}
   \_sdef{_spg:.}{\_egroups\_endslides}
   \_sdef{_spg:+}{\_egroups\_bgroup \_setilevel}
   \_let\_layersopen=\_egroup \_let\_layersclose\_bgroup
   \_bgroup
}
\_public \slideopen ;
```

When `\slideshow` is active then the destinations of internal hyperlinks cannot be duplicated to more “virtual” pages because hyperlink destinations have to be unique in the whole document.

The `\slideshow` creates boxes of typesetting material and copies them to more pages. So, we have to suppress creating destinations in these boxes. This is done in the `\_slidelinks` macro. We can move creating these destinations to the output routine. `\_sdestbox` is saved value of the original `\_destbox` which is redefined to do only `\addto\_destboxes{\_sdestbox[`⟨*label*⟩`]}`. All destinations saved to `\_destboxes` are created at the start of the next output routine in the `\_pagedest` macro. The output routine removes `\_destboxes`, so each destination is created only once.

Limitations of this solution: destinations are only at the start of the page, no at the real place where `\wlabel` was used. The first “virtual” page where `\wlabel` is used includes its destination. If you want to go to the final page of the partially uncovering ideas then use `\label[`⟨*label*⟩`]\wlabel{text}` in the last part of the page (before `\pg;`) o use `\pgref` instead `\ref`.

slides.opm

```
\_def\_slidelinks{%
   \_def \_destbox[##1]{\_edef\_tmp{\_noexpand\_sdestbox[##1]}%
      \_global\_ea\_addto\_ea\_destboxes\_ea{\_tmp}}%
   \_def \_pagedest {%
      \_hbox{\_def\_destheight{25pt}\_sdestbox[pg:\_the\_gpageno]\_destboxes}%
      \_nointerlineskip \_gdef\_destboxes{}%
   }%
   \_ifx \_dest\_destactive \_else \_let\_pagedest=\_relax \_fi
}
\_let\_sdestbox = \_destbox
\_def\_destboxes{}    % initial value of \_destboxes
\_let\_bibgl=\_global % \advance\bibnum must be global if they are at more pages
```

The `\_settinglayer` is used in the `\_layertext` macro to prevent printing “Duplicate label” warning when it is expanded. It is done by special value of `\_slideshook` (used by the `\label` macro). Moreower, the warning about illegal use of `\bib`, `\usebib` in `\layers` environment is activated.

slides.opm

```
\_def\_settinglayer{%
   \_def\_slideshook ##1##2{}%
   \_def\_bibB[##1]{\_nousebib}\_def\_usebib/##1 (##2) ##3 {\_nousebib}%
}
\_def\_nousebib{\_opwarning{Don't use \noexpand\bib nor \noexpand\usebib in \string\layers}}
```

Default `\layers` ⟨*num*⟩ macro (when `\slideshow` is not activated) is simple. It prints the ⟨*layered-text*⟩ with `\layernum=`⟨*num*⟩`+1` because we need the result after last layer is processed.

slides.opm

```
\_long\_def\_layers #1 #2\endlayers{\_par
   \_layersopen {\_layernum=\_numexpr#1+1\_relax #2\_endgraf}\_layersclose}
\_let\_layersopen=\_relax
\_let\_layersclose=\_relax

\_def\layers{\_layers}
```

We must to redefine `\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 `\pg;` macro, so we can reset the footnote counter by this macro.

slides.opm

```
\_def \_fnotenumpages {\_def\_fnotenum{\_the\_lfnotenum}\_pgfnotefalse
   \_def\_lfnotenumreset{\_global\_lfnotenum=0 }}
\_let \_lfnotenumreset=\_relax
\_public \fnotenumpages ;
```

2.36 Logos

logos.opm

```
\_codedecl \TeX {Logos TeX, LuaTeX, etc. <2025-01-22>} % 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.

logos.opm

```
\_protected\_def \_TeX {T\_kern-.1667em\_lower.5ex\_hbox{E}\_kern-.125emX\_ignoreslash}
\_protected\_def \_OpTeX {Op\_kern-.1em\_TeX}
\_protected\_def \_LuaTeX {Lua\_TeX}
\_protected\_def \_LuaHBTeX {Lua{\_setfontsize{mag.9}\_resizethefont HB}\_TeX}
\_protected\_def \_XeTeX {X\_kern-.125em\_phantom E%
   \_pdfsave\_rlap{\_pdfscale{-1}{1}\_lower.5ex\_hbox{E}}\_pdfrestore \_kern-.1667em \_TeX}

\_def\_ignoreslash {\_isnextchar/\_ignoreit{}}

\_public \TeX \OpTeX \LuaTeX \LuaHBTeX \XeTeX \ignoreslash ;
```

The `\ConTeXt` logo is implemented as in the ConTeXt format itself. The kerning between “Con” and “TeXt” is calculated by measuring the kerning between the letters “T” and “e”.

logos.opm

```
\_protected\_def \_ConTeXt{\_begingroup
    Con\_setbox0=\_hbox{T\_kern\_zo e}\_setbox1=\_hbox{Te}{\_kern\_dimexpr\_wd1 -\_wd0}%
    \_TeX t\_endgroup\_ignoreslash}

\_public \ConTeXt ;
```

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.

logos.opm

```
\_protected\_def \_LaTeX{\_tmpdim=.42ex L\_kern-.36em \_kern \_slantcorr % slant correction
  \_raise \_tmpdim \_hbox{\_thefontscale[710]A}%
  \_kern-.15em \_kern-\_slantcorr \_TeX}
\_def\_slantcorr{\_ea\_ignorept \_the\_fontdimen1\_font\_tmpdim}

\_public \LaTeX ;
```

`\OPmac`, `\CS` and `\csplain` logos.

logos.opm

```
\_def\_OPmac{\_leavevmode
   \_lower.2ex\_hbox{\_thefontscale[1400]O}\_kern-.86em P{\_em mac}\_ignoreslash}
\_def\_CS{$\_cal C$\_kern-.1667em\_lower.5ex\_hbox{$\_cal S$}\_ignoreslash}
\_def\_csplain{\_CS plain\_ignoreslash}

\_public \OPmac \CS \csplain ;
```

The expandable versions of logos used in Outlines need the expandable `\ignslash` (instead of the `\ignoreslash`).

logos.opm

```
\_def\_ignslash#1{\_ifx/#1\_else #1\_fi}
\_regmacro {}{}{% conversion for PDF outlines
   \_def\TeX{TeX\_ignslash}\_def\OpTeX{OpTeX\_ignslash}%
   \_def\LuaTeX{LuaTeX\_ignslash}\_def\XeTeX{XeTeX\_ignslash}%
   \_def\LaTeX{LaTeX\_ignslash}\_def\OPmac{OPmac\_ignslash}%
   \_def\ConTeXt{ConTeXt\_ignslash}%
   \_def\CS{CS}\_def\csplain{csplain\_ignslash}%
}
\_public \ignslash ;
```

2.37 Multilingual support

2.37.1 Lowercase, uppercase codes

All codes in Unicode table keep information about pairs lowecase-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` (upperase 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 exist then internal data are used. They follow to the end of the file `uni-lcuc.opm`.

uni-lcuc.opm

```
 3 \_wlog{Setting lccodes and uccodes for Unicode characters <2021-04-07>} % preloaded in format.
 4
 5 \_isfile{UnicodeData.txt}\_iftrue
 6 \_begingroup
 7    \_sdef{lc:Ll}#1#2#3#4{\_global\_lccode"#2="#2  \_global\_uccode"#2="0#3 }
 8    \_sdef{lc:Lu}#1#2#3#4{\_global\_lccode"#2="0#4 \_global\_uccode"#2="#2 }
 9    \_sdef{lc:Lo}#1#2#3#4{\_global\_lccode"#2="#2  \_global\_uccode"#2="#2 }
10    \_def\_pa#1;#2;#3;#4;#5;#6;#7;#8;#9;{\_ifx;#1;\_else\_ea\_pb\_fi{#1}{#3}}
11    \_def\_pb#1#2#3;#4;#5;#6;#7;#8 {\_csname lc:#2\_endcsname\_pc{#1}{#6}{#7}\_pa}
12    \_def\_pc#1#2#3{}      % ignored if the character hasn't Ll, Lu, nor Lo type
13    \_everyeof={;;;;;;;;;} % end of file
14    \_ea\_pa\_input UnicodeData.txt
15 \_endgroup \_endinput \_fi  % \endinput here, if UnicodeData.txt was loaded
16
17 % If UnicodeData.txt not found, we have internal copy here from csplain, 2014:
18
19 \_def\_tmp #1 #2 {\_ifx^#1^\_else
20    \_lccode"#1="#1
21    \_ifx.#2%
22       \_uccode"#1="#1
23    \_else
24       \_uccode"#2="#2
25       \_lccode"#2="#1
26       \_uccode"#1="#2
27    \_fi
28    \_ea \_tmp \_fi
29 }
30 \_tmp
31 00AA .
32 00B5 039C
33 00BA .
34 00E0 00C0
35 00E1 00C1
36 00E2 00C2
37 00E3 00C3
38 00E4 00C4
39 00E5 00C5
```

. . . etc., 15900 similar lines (see uni-lcuc.opm)

2.37.2 Multilingual phrases and quotation marks

languages.opm

```
\_codedecl \_mtext {Languages <2022-11-18>} % preloaded in format
```

Four words are generated by OpTEX macros: “Chapter”, “Table”, “Figure” and “Subject”. These phrases are generated depending on the current value of the `\language` register, if you use `\_mtext{`⟨*phrase-id*⟩`}`, specially `\_mtext{chap}`, `\_mtext{t}`, `\_mtext{f}` or `\_mtext{subj}`. If your macros generate more words then you can define such words by `\sdef{_mt:`⟨*phrase-id*⟩`:`⟨*lang-tag*⟩`}` where ⟨*phrase-id*⟩ is a label for the declared word and ⟨*lang-tag*⟩ is a language shortcut declared by `\_preplang`.

`languages.opm`

```
\_def\_mtext#1{\_trycs{_mt:#1:\_trycs{_lan:\_the\_language}{en}}
   {\_csname _mt:#1:en\_endcsname}}
```

We can declare such language-dependent words by

```
\_sdef{_mt:chap:en}{Chapter} \_sdef{_mt:chap:cs}{Kapitola}
\_sdef{_mt:t:en}{Table}      \_sdef{_mt:t:cs}{Tabulka}
```

etc. but we use more “compact” macro `\_langw` ⟨*lang-tag*⟩ ⟨*chapter*⟩ ⟨*table*⟩ ⟨*figure*⟩ ⟨*subject*⟩ for declaring them.

`languages.opm`

```
\_def \_langw #1 #2 #3 #4 #5 {%
   \_sdef{_mt:chap:#1}{#2}\_sdef{_mt:t:#1}{#3}\_sdef{_mt:f:#1}{#4}%
   \_sdef{_mt:subj:#1}{#5}%
}
```

More phrases are auto-generated in bibliography references. They are declared by
`\_langb` ⟨*lang-tag*⟩ `{`⟨*and*⟩`}` `{`⟨*et-al*⟩`}` `{`⟨*ed*⟩`}` `{`⟨*cit*⟩`}` `{`⟨*vol*⟩`}` `{`⟨*no*⟩`}` `{`⟨*pp*⟩`}` `{`⟨*p*⟩`}` `{`⟨*ed*⟩`}` `{`⟨*eds*⟩`}` `{`⟨*avail-from*⟩`}` `{`⟨*avali-to*⟩`}` `{`⟨*ba-thesis*⟩`}` `{`⟨*ma-thesis*⟩`}` `{`⟨*phd-thesis*⟩`}`. It is used similar way as the `\_langw` above. Both these macros are used in `lang-data.opm` file, see the end of section 2.37.3.

`languages.opm`

```
\_def\_langb#1 #2#3#4#5#6#7#8#9{\_def\_mbib##1##2{\_sdef{_mt:bib.##2:#1}{##1}}%
   \_mbib{#2}{and}\_mbib{#3}{etal}\_mbib{#4}{edition}\_mbib{#5}{citedate}\_mbib{#6}{volume}%
   \_mbib{#7}{number}\_mbib{#8}{prepages}\_mbib{#9}{postpages}\_langbA}
\_def\_langbA#1#2#3#4#5#6#7{\_mbib{#1}{editor}\_mbib{#2}{editors}\_mbib{#3}{available}%
   \_mbib{#4}{availablealso}\_mbib{#5}{bachthesis}\_mbib{#6}{masthesis}\_mbib{#7}{phdthesis}}
```

`\today` macro needs auto-generated words for each name of the month.
`\_monthw` ⟨*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}{\_mtext{m\_the\_month} \_the\_day, \_the\_year}
```

This example declares date format for English where ⟨*lang-tag*⟩ is `en`.

`languages.opm`

```
\_def \_monthw #1 #2 #3 #4 #5 #6 #7 {%
   \_sdef{_mt:m1:#1}{#2}\_sdef{_mt:m2:#1}{#3}\_sdef{_mt:m3:#1}{#4}%
   \_sdef{_mt:m4:#1}{#5}\_sdef{_mt:m5:#1}{#6}\_sdef{_mt:m6:#1}{#7}%
   \_monthwB #1
}
\_def \_monthwB #1 #2 #3 #4 #5 #6 #7 {%
   \_sdef{_mt:m7:#1}{#2}\_sdef{_mt:m8:#1}{#3}\_sdef{_mt:m9:#1}{#4}%
   \_sdef{_mt:m10:#1}{#5}\_sdef{_mt:m11:#1}{#6}\_sdef{_mt:m12:#1}{#7}%
}
\_def\_today{\_mtext{today}}
\_public \today ;
```

Quotes should be tagged by `\"`⟨*text*⟩`"` and `\'`⟨*text*⟩`'` if `\`⟨*iso-code*⟩`quotes` is declared at beginning of the document (for example `\enquotes`). If not, then the control sequences `\"` and `\'` are undefined. Remember, that they are used in another meaning when the `\oldaccents` command is used. The macros `\"` and `\'` are not defined as `\protected` because we need their expansion when `\outlines` are created. User can declare quotes by `\quoteschars`⟨*clqq*⟩⟨*crqq*⟩⟨*clq*⟩⟨*crq*⟩, where ⟨*clqq*⟩...⟨*crqq*⟩ are normal quotes and ⟨*clq*⟩...⟨*crq*⟩ are alternative quotes. or use `\altquotes` to swap between the meaning of these two types of quotes. `\enquotes`, `\csquotes`, `\frquotes`, `\dequotes`, `\skquotes` are defined here. Languages in general provide the `\quotes` declaration macro. It declares the quotation marks depending on the actual selected language. For example, `\eslang \quotes` declares Spanish language including its quotation marks used for `\"`⟨*text*⟩`"` and `\'`⟨*text*⟩`'`. The language-dependent quotation marks should be declared by `\_quotationmarks` ⟨*lang-tag*⟩ `{`⟨*clqq*⟩⟨*crqq*⟩⟨*clq*⟩⟨*crq*⟩`}` in the `lang-data.opm` file.

languages.opm

```
\_def \_enquotes {\_quoteschars ""‘’}
\_def \_csquotes {\_quoteschars „“‚‘}
\_def \_frquotes {\_quoteschars ""«»}
\_let \_dequotes = \_csquotes
\_let \_skquotes = \_csquotes

\_def \_quotes {\_trycs{_qt:\_trycs{_lan:\_the\_language}{en}}{\_enquotes}}
\_def \_quotationmarks #1 #2{\_sdef{_qt:#1}{\_quoteschars #2}}

\_public \quotes \enquotes \csquotes \frquotes \dequotes \skquotes ;
```

The `\quoteschars`⟨*lqq*⟩⟨*rqq*⟩⟨*lq*⟩⟨*rq*⟩ defines `\"` and `\"` as `\_qqA` in normal mode and as expadable macros in outline mode. We want to well process the common cases: ``\"`&`"`` or ``\"`{`"``. This is the reason why the quotes parameter is read in verbatim mode and retokenized again by `\scantextokens`. We want to allow to quote the quotes mark itself by ``\"{`"`}"``. This is the reason why the sub-verbatim mode is used when the first character is `{` in the parameter.
The `\"` is defined as `\_qqA\_qqB`⟨*lqq*⟩⟨*rqq*⟩ and `\'` as `\_qqA\_qqC`⟨*lq*⟩⟨*rq*⟩. The `\_qqA\_qqB`⟨*clqq*⟩⟨*crqq*⟩ runs `\_qqB`⟨*lqq*⟩⟨*rqq*⟩⟨*text*⟩`"`.
The `\_regquotes\""`⟨*L*⟩⟨*R*⟩ does `\def\"#1"{`⟨*L*⟩`#1`⟨*R*⟩`}` for outlines but the `"` separator is active (because `"` and `'` are active in `\pdfunidef`).

languages.opm

```
\_def \_quoteschars #1#2#3#4{\_def\_altquotes{\_quoteschars#3#4#1#2}\_public\altquotes;%
   \_protected\_def {\_qqA\_qqB#1#2}\_protected\_def {\_qqA\_qqC#3#4}%
   \_regmacro{}{}{\_regquotes"#1#2\_regquotes'#3#4}}

\_def\_qqA#1#2#3{\_bgroup\_setverb \_catcode`=10
   \_isnextchar\_bgroup{\_catcode`=1 \_catcode`=2 #1#2#3}{#1#2#3}}
\_def\_qqB#1#2#3"{\_egroup#1\_scantextokens{#3}#2}
\_def\_qqC#1#2#3'{\_egroup#1\_scantextokens{#3}#2}
\_def\_regquotes#1#2#3#4{\_bgroup \_lccode`~=`#2\_lowercase{\_egroup \_def#1##1~}{#3##1#4}}
```

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`.

languages.opm

```
\_def\_activequotes{\_let\_actqq=\_adef"{\_actqq}\_let\_actq=\_adef'{\_actq}%
   \_regmacro{}{}{\_adef"{}\_adef'{}}}

\_public \quoteschars \activequotes ;
```

2.37.3 Languages declaration

lang-decl.opm

```
\_codedecl \langlist {Languages declaration <2025-05-12>} % preloaded in format
```

`\_preplang` ⟨*lang-id*⟩ ⟨*LongName*⟩ ⟨*lang-tag*⟩ ⟨*hyph-tag*⟩ ⟨*lr-hyph*⟩ declares a new language. The parameters (separated by space) are

- ⟨*lang-id*⟩: language identifier. It should be derived from ISO 639-1 code but additional letters can be eventually added because ⟨*lang-id*⟩ must be used uniquely in the whole declaration list. The `\_preplang` macro creates the language switch `\_`⟨*lang-id*⟩`lang` and defines also `\`⟨*lang-id*⟩`lang` as a macro which expands to `\_`⟨*lang-id*⟩`lang`. For example, `\_preplang cs Czech ...` creates `\_cslang` as the language switch and defines `\def\cslang{\_cslang}`.
- ⟨*LongName*⟩: full name of the language.
- ⟨*lang-tag*⟩: language tag, which is used for setting language-dependent phrases and sorting data. If a language have two or more hyphenation patterns but a single phrases set, then we declare this language more than once with the same ⟨*lang-tag*⟩ but different ⟨*lang-hyph*⟩.
- ⟨*hyph-tag*⟩: a part of the file name where the hyphenation patterns are prepared in Unicode. The full file name is `hyph-`⟨*hyph-tag*⟩`.tex`. If ⟨*hyph-tag*⟩ is `{}` then no hyphenation patterns are loaded.
- ⟨*lr-hyph*⟩: two digits, they denote `\lefthyphenmin` and `\righthyphenmin` values.

`\_preplang` allocates a new internal number by `\newlanguage\_`⟨*lang-id*⟩`Patt` which will be bound to the hyphenation patterns. But the patterns nor other language data are not read at this moment. The `\_`⟨*lang-id*⟩`lang` is defined as `\_langinit`. When the `\_`⟨*lang-id*⟩`lang` switch is used firstly in a document then the language is initialized, i.e. hyphenation patterns and language-dependent data are read. The `\_`⟨*lang-id*⟩`lang` is re-defined itself after such initialization. `\_preplang` does also `\def\_ulan:`⟨*longname*⟩ `{`⟨*lang-id*⟩`}`, this is needed for the `\uselanguage` macro.

lang-decl.opm
```
\_def\_preplang #1 #2 #3 #4 #5#6{% lang-id LongName lang-tag hyph-tag lr-hyph
   \_ifcsname _#1lang\_endcsname \_else
      \_ea\_newlanguage\_csname _#1Patt\_endcsname
      \_xdef\_langlist{\_langlist\_space#1(#2)}%
   \_fi
   \_lowercase{\_sxdef{_ulan:#2}}{#1}%
   \_slet{_#1lang}{_relax}%
   \_sxdef {#1lang}{\_cs{_#1lang}}%
   \_sxdef {_#1lang}{\_noexpand\_langinit \_cs{_#1lang}#1(#2)#3[#4]#5#6}%
}
```

The `\_preplang` macro adds ⟨*lang-id*⟩(⟨*LongName*⟩) to the `\_langlist` macro which is accessible by `\langlist`. It can be used for reporting declared languages.

lang-decl.opm
```
\_def\langlist{\_langlist}
\_def\_langlist{en(USEnglish)}
```

All languages with hyphenation patterns provided by TEXlive are declared here. The language switches `\cslang`, `\sklang`, `\delang`, `\pllang` and many others are declared. You can declare more languages by `\_preplang` in your document, if you want.
The usage of `\_preplang` with ⟨*lang-id*⟩ already declared is allowed. The language is re-declared in this case. This can be used in your document before first usage of the `\`⟨*lang-id*⟩`lang` switch.

lang-decl.opm
```
%          lang-id LongName        lang-tag  hyph-tag       lr-hyph
\_preplang enus     USenglishmax    en        en-us            23
% Europe:
\_preplang engb     UKenglish       en        en-gb            23
\_preplang be       Belarusian      be        be               22
\_preplang bg       Bulgarian       bg        bg               22
\_preplang ca       Catalan         ca        ca               22
\_preplang hr       Croatian        hr        hr               22
\_preplang cs       Czech           cs        cs               23
\_preplang da       Danish          da        da               22
\_preplang nl       Dutch           nl        nl               22
\_preplang et       Estonian        et        et               23
\_preplang fi       Finnish         fi        fi               22
\_preplang fis      schoolFinnish   fi        fi-x-school      11
\_preplang fr       French          fr        fr               22
\_preplang de       nGerman         de        de-1996          22
\_preplang deo      oldGerman       de        de-1901          22
\_preplang gsw      swissGerman     de        de-ch-1901       22
\_preplang elm      monoGreek       el        el-monoton       11
\_preplang elp      Greek           el        el-polyton       11
\_preplang grc      ancientGreek    grc       grc              11
\_preplang hu       Hungarian       hu        hu               22
\_preplang is       Icelandic       is        is               22
\_preplang ga       Irish           ga        ga               23
\_preplang it       Italian         it        it               22
\_preplang la       Latin           la        la               22
\_preplang lac      classicLatin    la        la-x-classic     22
\_preplang lal      liturgicalLatin la        la-x-liturgic    22
\_preplang lv       Latvian         lv        lv               22
\_preplang lt       Lithuanian      lt        lt               22
\_preplang mk       Macedonian      mk        mk               22
\_preplang pl       Polish          pl        pl               22
\_preplang pt       Portuguese      pt        pt               23
\_preplang ro       Romanian        ro        ro               22
\_preplang rm       Romansh         rm        rm               22
\_preplang ru       Russian         ru        ru               22
\_preplang srl      Serbian         srlatn    sh-latn          22
\_preplang src      SerbianCyrl     srcyrl    sh-cyrl          22
\_preplang sk       Slovak          sk        sk               23
\_preplang sl       Slovenian       sl        sl               22
\_preplang es       Spanish         es        es               22
\_preplang sv       Swedish         sv        sv               22
```

```
\_preplang uk       Ukrainian       uk        uk                 22
\_preplang cy       Welsh           cy        cy                 23
% Others:
\_preplang af       Afrikaans       af        af                 12
\_preplang hy       Armenian        hy        hy                 12
\_preplang as       Assamese        as        as                 11
\_preplang eu       Basque          eu        eu                 22
\_preplang bn       Bengali         bn        bn                 11
\_preplang nb       Bokmal          nb        nb                 22
\_preplang cop      Coptic          cop       cop                11
\_preplang cu       churchslavonic  cu        cu                 12
\_preplang eo       Esperanto       eo        eo                 22
\_preplang ethi     Ethiopic        ethi      mul-ethi           11
\_preplang fur      Friulan         fur       fur                22
\_preplang gl       Galician        gl        gl                 22
\_preplang ka       Georgian        ka        ka                 12
\_preplang gu       Gujarati        gu        gu                 11
\_preplang hi       Hindi           hi        hi                 11
\_preplang id       Indonesian      id        id                 22
\_preplang ia       Interlingua     ia        ia                 22
\_preplang kn       Kannada         kn        kn                 11
\_preplang kmr      Kurmanji        kmr       kmr                22
\_preplang ml       Malayalam       ml        ml                 11
\_preplang mr       Marathi         mr        mr                 11
\_preplang mn       Mongolian       mn        mn-cyrl            22
\_preplang nn       Nynorsk         nn        nn                 22
\_preplang oc       Occitan         oc        oc                 22
\_preplang or       Oriya           or        or                 11
\_preplang pi       Pali            pi        pi                 12
\_preplang pa       Panjabi         pa        pa                 11
\_preplang pms      Piedmontese     pms       pms                22
\_preplang zh       Pinyin          zh        zh-latn-pinyin     11
\_preplang sa       Sanskrit        sa        sa                 13
\_preplang ta       Tamil           ta        ta                 11
\_preplang te       Telugu          te        te                 11
\_preplang th       Thai            th        th                 23
\_preplang tr       Turkish         tr        tr                 22
\_preplang tk       Turkmen         tk        tk                 22
\_preplang hsb      Uppersorbian    hsb       hsb                22

\_preplang he       Hebrew          he        {}                 00
```

`\_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.

lang-decl.opm

```
\_def\_preplangmore #1 #2{\_ea \_gdef \_csname _langspecific:#1\_endcsname{#2}}

\_preplangmore en   {\_nonfrenchspacing}
\_preplangmore enus {\_nonfrenchspacing}
\_def\_langdefault  {\_frenchspacing}
```

The `\_langreset` is processed before macros declared by `\_preplangmore` or before `\_langdefault`. If you set something for your language by `\_preplangmore` then use `\def\_langreset{`⟨*settings*⟩`}` in this code too in order to return default values for all other languages. See `cs` part of `lang-data.opm` file for an example.

lang-decl.opm

```
\_def\_langreset {}
```

The default `\language=0` is US-English with original hyphenation patterns preloaded in the format (see the end of section 2.10). We define `\_enlang` and `\enlang` switches. Note that if no language switch is used in the document then `\language=0` and US-English patterns are used, but `\nonfrenchspacing` isn't set.

lang-decl.opm

```
\_chardef\_enPatt=0
\_sdef{_lan:0}{en}
\_sdef{_ulan:usenglish}{en}
\_def\_enlang{\_uselang{en}\_enPatt23} % \lefthyph=2 \righthyph=3
\_def\enlang{\_enlang}
```

The list of declared languages are reported during format generation.

lang-decl.opm

```
\_message{Declared languages: \_langlist.
   Use \_stringlang-id>lang to initialize language,
   \_string\cslang\_space for example.}
```

Each language switch `\_`⟨*lang-id*⟩`lang` defined by `\_preplang` has its initial state `\_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.

lang-decl.opm

```
\_def\_langinit #1#2(#3)#4[#5]#6#7{% \_switch lang-id(LongName)lang-tag[hyph-file]lr-hyph
   \_sxdef{_lan:\_ea\_the\_csname _#2Patt\_endcsname}{#4}%
   \_begingroup \_setbox0=\_vbox{% we don't want spaces in horizontal mode
      \_setctable\_optexcatcodes
      % loading patterns:
      \_language=\cs{_#2Patt}\_relax
      \_ifx^#5^\_else
         \_wlog{Loading hyphenation for #3: \_string\language=\_the\_language\_space(#5)}%
         \_let\patterns=\_patterns \_let\hyphenation=\_hyphenation \_def\message##1{}%
         \_isfile {hyph-#5}\_iftrue \_input{hyph-#5}%
         \_else \_opwarning{No hyph. patterns #5 for #3, missing package?}\_fi
      \_fi
      % loading language data:
      \_langinput{#4}%
   }\_endgroup
   \xdef#1{\noexpand\_uselang{#2}\_csname _#2Patt\_endcsname #6#7}%
   #1% do language switch
}
```

`\_uselang{`⟨*lang-id*⟩`}\_`⟨*lang-id*⟩`Patt` ⟨*pre-hyph*⟩⟨*post-hyph*⟩ is used as "normal job" of the switch. It sets `\language`, `\lefthyphenmin`, `\righthyphenmin`. Finally, it runs data from `\_preplangmore` or runs `\_langdefault`.

lang-decl.opm

```
\_def\_uselang#1#2#3#4{\_language=#2\_lefthyphenmin=#3\_righthyphenmin=#4\_relax
   \_langreset \_def\_langreset{}\_trycs{_langspecific:#1}{\_langdefault}%
}
```

The `\uselanguage` `{`⟨*LongName*⟩`}` macro is defined here (for compatibility with e-plain users). Its parameter is case insensitive.

lang-decl.opm

```
\_def\_uselanguage#1{\_def\_tmp{#1}\_lowercase{\_cs{_\_trycs{_ulan:#1}{0x}lang}}}
\_sdef{_0xlang}{\_opwarning{\_string\uselanguage{\_tmp}: Unknown language name, ignored}}
\_public \uselanguage ;
```

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{`⟨*lang-tag*⟩`}`. First few lines of the file looks like:

lang-data.opm

```
\_codedecl \_langdata {Language dependent data <2025-05-12>} % only en, cs preloaded in format

\_langdata en {English} % -----------------------------------------------
\_langw en  Chapter      Table       Figure       Subject
\_langb en  {, and }  { et al.} {ed.} {cit.~} {Vol.~} {No.~} {pp.~} {~p.} {,~ed.} {,~eds.}
            {Available from } {Available also from }
            {Bachelor's Thesis} {Master's Thesis} {Ph.D. Thesis}
\_monthw en January February March April May June
            July August September October November December
\_sdef{_mt:today:en}{\_mtext{m\_the\_month} \_the\_day, \_the\_year}
\_quotationmarks en {""‘’}

%\_let \_sortingdataen =  \_sortingdatalatin % set already, see section 2.33, makeindex.opm
%\_let \_ignoredcharsen = \_ignoredcharsgeneric
%\_def \_compoundcharsen {}

\_langdata cs {Czech}   % ------------------------------------------------
%           Chapter      Table       Figure       Subject
\_langw cs  Kapitola     Tabulka     Obrázek      Věc
%           {, and }  { et al.} {ed.} {cit.~} {Vol.~} {No.~} {pp.~} {~p.} {,~ed.} {,~eds.}
%           {Available from } {Available also from }
%           {Bachelor's Thesis} {Master's Thesis} {Ph.D. Thesis}
\_langb cs  { a } { a~kol.} {vyd.} {vid.~} {ročník~} {č.~} {s.~} {~s.} {,~editor} {,~editoři}
            {Dostupné na } {Dostupné též na }
            {Bakalářská práce} {Diplomová práce} {Disertační práce}
%           January February March April May June
%           July August September October November December
\_monthw cs ledna února března dubna května června
            července srpna září října listopadu prosince
\_sdef{_mt:today:cs}{\_the\_day.~\_mtext{m\_the\_month} \_the\_year} % date format
\_quotationmarks cs {„",‘}
\_preplangmore   cs {\_frenchspacing \_postexhyphenchar=`
                     \_def\_langreset{\_postexhyphenchar=0 }}

\_let \_sortingdatacs  = \_sortingdatalatin
\_let \_ignoredcharscs = \_ignoredcharsgeneric
\_def \_compoundcharscs {ch:^^T Ch:^^U CH:^^V} % see \_compoundchars in section 2.33


\_langdata de {German}  % ------------------------------------------------
\_langw de  Kapitel      Tabelle     Abbildung    Betreff
\_quotationmarks de {„",‘}
%todo
\_let \_sortingdatade  = \_sortingdatalatin
\_let \_ignoredcharsde = \_ignoredcharsgeneric
\_def \_compoundcharsde {ß:ss}
\_def \_xcompoundcharsde {} % ß is interpreted in second pass of sorting
```

...etc. (see lang-data.opm)

There are analogical declaration for more languages here. Unfortunately, this file is far for completeness. I welcome you send me a part of declaration for your language.

If your language is missing in this file then a warning is reported during language initialization. You can create your private declaration in your macros (analogical as in the `lang-data.opm` file but without the `\_langdata` prefix). Then you will want to remove the warning about missing data. This can be done by `\nolanginput{`⟨*lang-tag*⟩`}` given before initialization of your language.

The whole file `lang-data.opm` is not preloaded in the format because I suppose a plenty languages here and I don't want to waste the TeX memory by these declarations. Each part of this file prefixed by `\_langdata` ⟨*lang-tag*⟩ `{`⟨*LongName*⟩`}` is read separately when `\_langinput{`⟨*lang-tag*⟩`}` is used. And it is used in the `\_langinit` macro (i.e. when the language is initialized), so the appropriate part of this file is read automatically on demand.

If the part of the `lang-data.opm` concerned by ⟨*lang-tag*⟩ is read already then `\_li:`⟨*lang-tag*⟩ is set to `R` and we don't read this part of the file again.

lang-decl.opm

```
\_def\_langinput #1{%
   \_unless \_ifcsname _li:#1\_endcsname
      \_bgroup
          \_edef\_tmp{\_noexpand\_langdata #1 }\_everyeof\_ea{\_tmp{}}%
          \_long \_ea\_def \_ea\_tmp \_ea##\_ea1\_tmp{\_readlangdata{#1}}%
          \_globaldefs=1
          \_ea\_tmp \_input{lang-data.opm}%
          \_ea\_glet \_csname _li:#1\_endcsname R%
      \_egroup
   \_fi
}
\_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 ;
```

Data of two preferred languages are preloaded in the format:

`lang-decl.opm`

```
\_langinput{en} \_langinput{cs}
```

2.38 Other macros

Miscellaneous macros are here.

`others.opm`

```
\_codedecl \uv {Miscenaleous <2024-11-21>} % preloaded in format
```

`\useOpTeX` and `\useoptex` are declared as `\relax`.

`others.opm`

```
\_let \useOpTeX = \_relax   \_let \useoptex = \_relax
```

The `\lastpage` and `\totalpages` get the information from the `\_currpage`. The `\_Xpage` from `.ref` file sets the `\_currpage`.

`others.opm`

```
\_def\_totalpages {\_openref\_ea\_ignoresecond\_currpage}
\_def\_lastpage   {\_openref\_ea\_usesecond\_currpage}
\_def\_currpage {{0}{?}}
\_public \lastpage \totalpages ;
```

We need `\uv`, `\clqq`, `\crqq`, `\flqq`, `\frqq`, `\uslang`, `\ehyph` `\chyph`, `\shyph`, for backward compatibility with CSplain. Codes are set according to Unicode because we are using Czech only in Unicode when LuaTeX is used.

`others.opm`

```

% for compatibility with csplain:

\_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 `\letfont` was used in CSplain instead of `\fontlet`.

`others.opm`

```
\_let \letfont = \_fontlet
```

Non-breaking space in Unicode.

`others.opm`

```
\let ^^a0=~
```

Old macro packages need these funny control sequences. We don't use them in new macros.

others.opm

```
\_catcode`=11
\_let\z@=\_zo  \_let\z@skip=\_zoskip
\_newdimen\p@ \p@=1pt
\_toksdef\toks@=0
\_let\voidb@x=\_voidbox
\_chardefne=1 \_chardef\tw@=2 \_chardef\thr@@=3 \_chardef\sixt@@n=16
\_mathchardefm=1000 \_mathchardefM=10000 \_mathchardefMM=20000
\_countdef\m@ne=22 \m@ne=-1
\_chardefcclv=255 \_mathchardefcclvi=256
\_skipdef\skip@=0
\_dimendef\dimen@=0  \_dimendef\dimen@i=1
\_dimendef\dimen@ii=2
\_countdef\count@=255
\_def\m@th{\_mathsurround\z@}
\_def\o@lign{\_lineskiplimit\z@ \_oalign}
\_def\n@space{\_nulldelimiterspace\z@ \m@th}
\_newdimen\p@renwd \p@renwd=8.75pt
\_def\alloc@#1#2#3#4#5{\_allocator#5{\_csstring#2}#3}
\_catcode`=12
```

We don't want to read `opmac.tex` unless `\input opmac` is specified.

others.opm

```
\_def\OPmacversion{OpTeX}
```

We allow empty lines in math formulae. It is more comfortable.

others.opm

```
\_suppressmathparerror = 1
```

Lorem ipsum can be printed by `\lipsum`[⟨*range*⟩] or `\lorem`[⟨*range*⟩], for example `\lipsum[3]` or `\lipsum[112-121]`, max=150.

First usage of `\lipsum` reads the LaTeX file `lipsum.ltd.tex` by `\_lipsumload` and prints the selected paragraph(s). Next usages of `\lipsum` prints the selected paragraph(s) from memory. `\lipsum` is fully expandable.

`\lipsum` adds `\_par` after each printed paragraph. If you don't need such `\_par` here, use `\lipsumtext`[⟨*number*⟩] or `\lipsum`[⟨*number*⟩.] (i.e. dot after the parameter). The first case prints the paragraph ⟨*number*⟩ without the final `\_par` and the second case prints only first sentence from the paragraph ⟨*number*⟩ using `\_lipsumdot`.

others.opm

```
\_newbox\_nonebox
\_def\_lipsumtext[#1]{\_lipsumload\_cs{_lip:#1}}
\_def\_lipsum[#1]{\_lipsumA #1.]{#1}}
\_def\_lipsumA #1.#2]#3{\_ifx^#2^\_lipsumB #1\_empty-\_empty\_fin \_else \_lipsumdot[#1].\_fi}
\_def\_lipsumB #1-#2\_empty#3\_fin{%
   \_fornum #1..\_ifx^#2^#1\_else#2\_fi \_do {\_lipsumtext[##1]\_par}}
\_def\_lipsumfile{lipsum.ltd.tex}
\_def\_lipsumload{\_beglocalcontrol
   {\_setbox\_nonebox=\_vpack{\_tmpnum=0 % vertical mode during \input lipsum.ltd.tex
      \_def\ProvidesFile##1[##2]{}%
      \_def\SetLipsumLanguage##1{}%
      \_def\NewLipsumPar{\_incr\_tmpnum \_sxdef{_lip:\_the\_tmpnum}}%
      \_opinput\_lipsumfile
      \_glet\_lipsumload=\_empty
   }}%
   \_endlocalcontrol}
\_def\_lipsumdot[#1]{\_lipsumload \_ea\_ea\_ea \_lipsumdotA \_csname _lip:#1\_endcsname.\_fin}
\_def\_lipsumdotA #1.#2\_fin {#1}

\_public \lipsum \lipsumtext ;
\_let \lorem=\lipsum
```

Selected macros from OpTeX tricks are registered using `\_regtrick`⟨*cs-name*⟩. The ⟨*cs-name*⟩ is defined as `\loadtrick` ⟨*cs-name*⟩ ⟨*cs-name*⟩. When a user runs such a registered ⟨*cs-name*⟩ then `\loadtrick` ⟨*cs-name*⟩ reads the appropriate code from the file `optex-tricks.opm` and the ⟨*cs-name*⟩ is redefined. Finally, ⟨*cs-name*⟩ is run again.
The `optex-tricks.opm` file includes blocks started by `\_trick` followed by the declared ⟨*cs-names*⟩ followed by semicolon followed by the code with declarations of ⟨*cs-names*⟩ itself. The next `\_trick` does `\endpinput` of the file. The file is read inside temporary `\vbox` with `\globaldefs=1` because it can be

read inside horizontal mode and/or inside a group. The `optextrick` name space is used during reading the code from the file. Only registered control sequences are re-defined directly in user name space.
You can load a code chunk by `\loadtrick` ⟨*cs-name*⟩. This command doesn't run the ⟨*cs-name*⟩, only loads the appropriate code. It should be usable if you want to load the code before the first usage of the ⟨*cs-name*⟩.

others.opm

```
\_def\_regtrick#1{\_ifx#1\_undefined\_def#1{\_loadtrickD#1}\_else\_badtrick\_fi}
\_def\_loadtrickD#1{\_loadtrick#1#1}
\_def\_loadtrick#1{\_beglocalcontrol
   \_resetnamespace{optextrick}\_setctable\_optexcatcodes
   \_savecatcodetable\_tmpcatcodes \_catcodetable\_tmpcatcodes
   \_long\_def\_loadtrickA ##1\_trick##2#1##3;{##1}%
   \_wlog{Loading trick macros for \_string#1}%
   \_setbox\_nonebox=\_vpack{\_let=\_trickprefix \_def\_trickseq{#1}%
      \_globaldefs=1 \_ea\_loadtrickA \_input {optex-tricks.opm}}%
   \_restorectable \_endnamespace
   \_endlocalcontrol
}
\_def\_trick #1;{\_endinput}
\_public \loadtrick ;

\_xargs \_regtrick \begfile \createfile \beglua \begLUA \logginglua
   \sethours \setminutes \setseconds \setweekday \showpglists \shownodes \runsystem
   \directoutput \algol \algolkw \algolcap \algolnum \makeLOA \setaca \scaleto \scaletof
   \ttlineref \lref \easylist \keepstyle \fcread \shadedframe \roundframe \cancel \ignoreinspic
   \keystroke \colortab \crx \crtop \crbot \crmid \longtable \vcent \vbot \tnote
   \tabnodes \tablebefore \framedblocks \twoblocks \pstart \settabs \import \incrpp \ispageodd
   \iniseccc \seccc \makeLOF \makeLOT \captionF \captionT \correctvsize \pgforeground
   \onlyifnew \thedimen \rebox \leftfill \rightfill \lrfill \directchar \ereplstring
   \xreplstring \replmacro \tdnum \clipbox \ismatch \isinmacro \mathinexpr \xparshape
   \csvread \csvdecl \dbcreate \inode \enode \execedges \showtree \jsonread \jsoncnv
   \treedata \tracingtreedata \treenodes ;
\_sdef{_item:m}{\_loadtrick{\style m}\_cs{_item:m}}
```

The `\:` is used as a prefix before `\def` etc. in the file `optex-tricks.opm` if the declared control sequence is from user namespace. It checks if a user have defined it before the trick is loaded and prints warning in this case. The `\:` is set to `\_trickprefix` in the group before trick loading. If a trick includes only one autoloaded control sequence then it needs not to be prefixed by `\:` because if a user redefines it then the usage of it doesn't run autoloading.

others.opm

```
\_def\_trickprefix #1#2{\_ifx#2\_undefined \_else \_def\_tmp{\_loadtrickD#2}\_ifx\_tmp#2\_else
   \_opwarning{trick by \_ea\_string\_trickseq\_space loaded: \_string#2 redefined}\_fi\_fi #1#2}
```

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{...}`.

others.opm

```
\_ifx\_partokenname\_undefined % LuaTeX 1.13 or older:

   \_def\_incaption {\_bgroup
      \_ifcsname _\_tmpa num\_endcsname \_ea\_incr \_csname _\_tmpa num\_endcsname
      \_else \_opwarning{Unknown caption /\_tmpa}\_fi
      \_edef\_thecapnum {\_csname _the\_tmpa num\_endcsname}%
      \_edef\_thecaptitle{\_mtext{\_tmpa}}%
      \_ea\_the \_csname _everycaption\_tmpa\_endcsname
      \_def\_par{\_nbpar\_egroup}\_let\par=\_par
      \_cs{_printcaption\_tmpa}%
   }
   \_def\_boxlines{%
      \_def\_boxlinesE{\_ifhmode\_egroup\_empty\_fi}%
      \_def\_nl{\_boxlinesE}%
      \_bgroup \_lccode`=`^M\_lowercase{\_egroup\_let~}\_boxlinesE
      \_everypar{\_setbox0=\_lastbox\_endgraf
         \_hbox\_bgroup \_catcode`^M=13 \_let\par=\_nl \_aftergroup\_boxlinesC}%
   }
   \_def\_letter{
      \_def\_address{\_vtop\_bgroup\_boxlines \_parskip=0pt \_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
   }
   \_def\_printverbline#1{\_putttpenalty \_indent \_printverblinenum \_kern\_ttshift #1\par}
   \_public \boxlines \letter ;

\_else % LuaTeX 1.14 or newer:
```

We set `\partokenneame` 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. Se section 2.2.1 for more details about the name space concept.
Moreover, we set `\partokencontext` to one in order to the `\_par` token is inserted not only at empty lines, but also at the end of `\vbox`, `\vtop` and `\vcenter` if horizontal mode is opened here. This differs from default TEX behavior where horizontal mode is closed in these cases without inserting par token.
We set `\_partokenset` to defined value 1 in order to the macro programmer can easily check these settings in OpTEX format by `\ifx\_partokenset\undefined ... \else ...\fi`.

others.opm

```
   \_partokenname\_par
   \_partokencontext=1
   \_let\_partokenset=1
\_fi
```

2.39 Lua code embedded to the format

The file `optex.lua` is loaded into the format in `optex.ini` as byte-code and initialized by `\everyjob`, see section 2.1.

The file implements part of the functionality from `luatexbase` namespace, nowadays defined by LATEX kernel. `luatexbase` deals with modules, allocators, and callback management. Callback management is a nice extension and is actually used in OpTEX. Other functions are defined more or less just to suit luaotfload's use.

The allocations are declared in subsection 2.39.2, callbacks are implemented in subsection 2.39.3 and handling with colors can be found in the subsection 2.39.6.

Local imports of standard Lua or LuaTEX functions and constants are the first thing we do. They mostly serve one of several purposes:

- shorthands for commonly used functions,
- speed of access for commonly executed functions,
- common constants to be used consistently,
- use of local definition that can't be externally overwritten.

optex.lua

```

local fmt = string.format

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 kern_id = node_id("kern")
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 setattribute = direct.set_attribute
local insertbefore = direct.insert_before
local copy = direct.copy
local traverse = direct.traverse
local one_bp = tex.sp("1bp")

local tex_print = tex.print
local tex_setbox = tex.setbox
local tex_getcount = tex.getcount
local tex_setcount = tex.setcount
local token_scanint = token.scan_int
local token_scanlist = token.scan_list
local token_setmacro = token.set_macro
```

2.39.1 General

Define namespace where "public" OpTeX functions will be added.

```

local optex = _ENV.optex or {}
_ENV.optex = optex

```

Error function used by following functions for critical errors.

```
local function err(...)
    local message = fmt(...)
    error("\nerror: "..message.."\n")
end
```

For a `\chardef`'d, `\countdef`'d, etc., csname return corresponding register number. The responsibility of providing a `\XXdef`'d name is on the caller.

```
local function registernumber(name)
    return token.create(name).index
end
_ENV.registernumber = registernumber
optex.registernumber = registernumber
```

MD5 hash of given file.

```
function optex.mdfive(file)
    local fh = io.open(file, "rb")
    if fh then
        local data = fh:read("*a")
        fh:close()
        tex_print(md5.sumhexa(data))
   end
end

```

The `optex.raw_ht()` function measures the height plus depth of given `\vbox` saved by `\setbox`. It solves the "dimension too large" problem with big boxes. It is used in the `\_rawht` macro.

```

function optex.raw_ht()
    local nod = tex.box[token.scan_int()]       -- read head node of the given box
    local ht = 0
    if nod ~= nil and nod.id == vlist_id then
        for n in node.traverse(nod.head) do
            if n.id == hlist_id or n.id == vlist_id or n.id == rule_id then
                ht = ht + n.height + n.depth
            elseif n.id == glue_id then
                ht = ht + n.width
            elseif n.id == kern_id then
                ht = ht + n.kern
            end
        end
    end
    tex_print(fmt('%.0f', ht/65536))
end

```

2.39.2 Allocators

```
local alloc = _ENV.alloc or {}
_ENV.alloc = alloc
```

An attribute allocator in Lua that cooperates with normal OpTEX allocator.

```
local attributes = {}
function alloc.new_attribute(name)
    local cnt = tex.count["_attributealloc"] + 1
    if cnt > 65534 then
        tex.error("No room for a new attribute")
    else
        tex.setcount("global", "_attributealloc", cnt)
        texio.write_nl("log", '"'..name..'"=\\attribute'..tostring(cnt))
        attributes[name] = cnt
        return cnt
    end
end
```

Allocator for Lua functions ("pseudoprimitives"). It passes variadic arguments ("`...`") like `"global"` to `token.set_lua`.

```
local function_table = lua.get_functions_table()
local function define_lua_command(csname, fn, ...)
    local luafnalloc = #function_table + 1
    token.set_lua(csname, luafnalloc, ...) -- WARNING: needs LuaTeX 1.08 (2019) or newer
    function_table[luafnalloc] = fn
end
_ENV.define_lua_command = define_lua_command
optex.define_lua_command = define_lua_command
```

2.39.3 Callbacks

```
local callback = _ENV.callback or {}
_ENV.callback = callback
```

Save `callback.register` function for internal use.

```
local callback_register = callback.register
function callback.register(name, fn)
    err("direct registering of callbacks is forbidden, use 'callback.add_to_callback'")
end
```

Table with lists of functions for different callbacks.

```
local callback_functions = {}
```

Table that maps callback name to a list of descriptions of its added functions. The order corresponds with `callback_functions`.

```
local callback_description = {}
```

Table used to differentiate user callbacks from standard callbacks. Contains user callbacks as keys.

```
local user_callbacks = {}
```

Table containing default functions for callbacks, which are called if either a user created callback is defined, but doesn't have added functions or for standard callbacks that are "extended" (see `mlist_to_hlist` and its pre/post filters below).

```
local default_functions = {}
```

Table that maps standard (and later user) callback names to their types.

```
local callback_types = {
    -- file discovery
    find_read_file     = "exclusive",
    find_write_file    = "exclusive",
    find_font_file     = "data",
    find_output_file   = "data",
    find_format_file   = "data",
    find_vf_file       = "data",
    find_map_file      = "data",
    find_enc_file      = "data",
    find_pk_file       = "data",
    find_data_file     = "data",
    find_opentype_file = "data",
    find_truetype_file = "data",
    find_type1_file    = "data",
    find_image_file    = "data",

    open_read_file     = "exclusive",
    read_font_file     = "exclusive",
    read_vf_file       = "exclusive",
    read_map_file      = "exclusive",
    read_enc_file      = "exclusive",
    read_pk_file       = "exclusive",
    read_data_file     = "exclusive",
    read_truetype_file = "exclusive",
    read_type1_file    = "exclusive",
    read_opentype_file = "exclusive",

    -- data processing
    process_input_buffer  = "data",
    process_output_buffer = "data",
    process_jobname       = "data",
    input_level_string    = "data",

    -- node list processing
    contribute_filter      = "simple",
    buildpage_filter       = "simple",
    build_page_insert      = "exclusive",
    pre_linebreak_filter   = "list",
    linebreak_filter       = "exclusive",
    append_to_vlist_filter = "exclusive",
    post_linebreak_filter  = "reverselist",
    hpack_filter           = "list",
    vpack_filter           = "list",
    hpack_quality          = "list",
    vpack_quality          = "list",
    process_rule           = "exclusive",
    pre_output_filter      = "list",
    hyphenate              = "simple",
    ligaturing             = "simple",
    kerning                = "simple",
    insert_local_par       = "simple",
    mlist_to_hlist         = "exclusive",

    -- information reporting
    pre_dump             = "simple",
    start_run            = "simple",
    stop_run             = "simple",
    start_page_number    = "simple",
    stop_page_number     = "simple",
    show_error_hook      = "simple",
    show_error_message   = "simple",
    show_lua_error_hook  = "simple",
    start_file           = "simple",
    stop_file            = "simple",
    call_edit            = "simple",
    finish_synctex       = "simple",
    wrapup_run           = "simple",

    -- pdf related
    finish_pdffile            = "data",
    finish_pdfpage            = "data",
    page_order_index          = "data",
    process_pdf_image_content = "data",

    -- font related
    define_font     = "exclusive",
    glyph_not_found = "exclusive",
    glyph_info      = "exclusive",

    -- undocumented
    glyph_stream_provider = "exclusive",
    provide_charproc_data = "exclusive",
}
```

Return a list containing descriptions of added callback functions for specific callback.

```
function callback.callback_descriptions(name)
    return callback_description[name] or {}
end

local valid_callback_types = {
    exclusive = true,
    simple = true,
    data = true,
    list = true,
    reverselist = true,
}
```

Create a user callback that can only be called manually using `call_callback`. A default function is only needed by "exclusive" callbacks.

```
function callback.create_callback(name, cbtype, default)
    if callback_types[name] then
        err("cannot create callback '%s' - it already exists", name)
    elseif not valid_callback_types[cbtype] then
        err("cannot create callback '%s' with invalid callback type '%s'", name, cbtype)
    elseif cbtype == "exclusive" and not default then
        err("unable to create exclusive callback '%s', default function is required", name)
    end

    callback_types[name] = cbtype
    default_functions[name] = default or nil
    user_callbacks[name] = true
end
```

Add a function to the list of functions executed when callback is called. For standard luatex callback a proxy function that calls our machinery is registered as the real callback function. This doesn't happen for user callbacks, that are called manually by user using `call_callback` or for standard callbacks that have default functions – like `mlist_to_hlist` (see below).

```
local call_callback
function callback.add_to_callback(name, fn, description)
    if user_callbacks[name] or callback_functions[name] or default_functions[name] then
        -- either:
        --  a) user callback - no need to register anything
        --  b) standard callback that has already been registered
        --  c) standard callback with default function registered separately
        --     (mlist_to_hlist)
    elseif callback_types[name] then
```

```
        -- This is a standard luatex callback with first function being added,
        -- register a proxy function as a real callback. Assert, so we know
        -- when things break, like when callbacks get redefined by future
        -- luatex.
        callback_register(name, function(...)
            return call_callback(name, ...)
        end)
    else
        err("cannot add '%s' to callback '%s' - no such callback exists", description, name)
    end

    if not description or description == "" then
        err("missing description when adding a callback to '%s'", name)
    end

    local prev_descriptions = callback_description[name] or {}
    for _, desc in ipairs(prev_descriptions) do
        if desc == description then
            err("for callback '%s' there already is '%s' added", name, description)
        end
    end

    if callback_types[name] == "exclusive" and #prev_descriptions == 1 then
        err("can't register '%s' as callback '%s' - exclusive callback occupied by '%s'",
            description, name, prev_descriptions[1])
    end

    if type(fn) ~= "function" then
        err("expected Lua function to be added as '%s' for callback '%s'", description, name)
    end

    -- add function to callback list for this callback
    callback_functions[name] = callback_functions[name] or {}
    table.insert(callback_functions[name], fn)

    -- add description to description list
    callback_description[name] = callback_description[name] or {}
    table.insert(callback_description[name], description)
end
```

Remove a function from the list of functions executed when callback is called. If last function in the list is removed delete the list entirely.

```
function callback.remove_from_callback(name, description)
    local descriptions = callback_description[name]
    local index
    for i, desc in ipairs(descriptions) do
        if desc == description then
            index = i
            break
        end
    end

    if not index then
        err("can't remove '%s' from callback '%s': not found", description, name)
    end

    table.remove(descriptions, index)
    local fn = table.remove(callback_functions[name], index)

    if #descriptions == 0 then
        -- Delete the list entirely to allow easy checking of "truthiness".
        callback_functions[name] = nil

        if not user_callbacks[name] and not default_functions[name] then
            -- this is a standard callback with no added functions and no
            -- default function (i.e. not mlist_to_hlist), restore standard
            -- behaviour by unregistering.
            callback_register(name, nil)
        end
```

```lua
    end

    return fn, description
end
```

helper iterator generator for iterating over reverselist callback functions

```lua
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.

```lua
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 '%s' - no such callback exists", name)
    elseif cbtype == "exclusive" then
        -- only one function, at least default function is guaranteed by
        -- create_callback
        return functions[1](...)
    elseif cbtype == "simple" then
        -- call all functions one after another, no passing of data
        for _, fn in ipairs(functions) do
            fn(...)
        end
        return
    elseif cbtype == "data" then
        -- pass data (first argument) from one function to other, while keeping
        -- other arguments
        local data = (...)
        for _, fn in ipairs(functions) do
            data = fn(data, select(2, ...))
        end
        return data
    end

    -- list and reverselist are like data, but "true" keeps data (head node)
    -- unchanged and "false" ends the chain immediately
    local iter
    if cbtype == "list" then
        iter = ipairs
    elseif cbtype == "reverselist" then
        iter = reverse_ipairs
    end

    local head = (...)
    local new_head
    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
        end
    end
    return 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
    else
        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
    end
    return new_head
end)
```

For preprocessing boxes just before shipout we define custom callback. This is used for coloring based on attributes. There is however a challenge - how to call this callback? We could redefine `\shipout` and `\pdfxform` (which both run `ship_out` procedure internally), but they would lose their primitive meaning – i.e. `\immediate` wouldn’t work with `\pdfxform`. The compromise is to require anyone to run `\_preshipout`⟨*destination box number*⟩⟨*box specification*⟩ just before `\shipout` or `\pdfxform` if they want to call `pre_shipout_filter` (and achieve colors and possibly more).

```
callback.create_callback("pre_shipout_filter", "list")

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 PDF object utilities

The PDF format defines various kinds of “objects”: numbers, names, arrays, dictionaries. A PDF document is mostly a tree of these objects (i.e. objects contain other objects), with either direct (“in-place”) or indirect references.

These objects are saved in the PDF file in a serialized form, often compressed. While LuaTEX takes care of the compression, and most of the structure of the PDF format, sometimes we want to insert our own PDF objects - and there are places where LuaTEX allows us to use our own objects to insert anything, but it already expects a serialized form of the objects. The serialized form for some kinds of objects has various formatting and escaping rules, and is inconvenient to produce in TEX especially when some of the e.g. names and strings are *user input*.

Here we define a couple of utilities for both Lua and TEX that aid with representation and serialization of objects. The functions are exported, but may still evolve in incompatible ways.

A reference for the serialization is the “Syntax” section of the PDF specification[3].

`\pdfname{`⟨*name*⟩`}` serializes a “PDF name” to bytes.

```
local function name_escape(char)
    return fmt("#%02X", char:byte())
end
```

```
local function pdf_name(name)
    -- Delimiters (i.e. "()<>[]{}/%") and control characters (0x00-0x1f, and 0x7f),
    -- space (" ", 0x20) as well as number sign ("#", 0x23) characters have to be escaped
    return "/" .. name:gsub("[%(%)<>%[%]{}/%%%c #]", name_escape)
end
optex.pdf_name = pdf_name
define_lua_command("_pdfname", function()
    tex_print(-2, pdf_name(token.scan_string()))
end)
```

`\pdfstring{`⟨*string*⟩`}` serializes a PDF string to bytes.

```
local function pdf_string(str)
    local out = {"<FEFF"}
    for _, c in utf8.codes(str) do
        if c < 0x10000 then
            out[#out+1] = fmt("%04X", c)
        else
            c = c - 0x10000
            local high = bit32.rshift(c, 10) + 0xD800
            local low = bit32.band(c, 0x3FF) + 0xDC00
            out[#out+1] = fmt("%04X%04X", high, low)
        end
    end
    out[#out+1] = ">"
    return table.concat(out)
end
optex.pdf_string = pdf_string
define_lua_command("_pdfstring", function()
    tex_print(-2, pdf_string(token.scan_string()))
end)
```

```
local function pdf_ref(id)
    return fmt("%d 0 R", id)
end
optex.pdf_ref = pdf_ref
```

---

[3] https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/PDF32000_2008.pdf

```
local pdfdict_mt = {
    __tostring = function(dict)
        local out = {"<<"}
        for k, v in pairs(dict) do
            out[#out+1] = fmt("%s %s", pdf_name(k), tostring(v))
        end
        out[#out+1] = ">>"
        return table.concat(out, "\n")
    end,
}
local function pdf_dict(t)
    return setmetatable(t or {}, pdfdict_mt)
end
optex.pdf_dict = pdf_dict
```

```
tex.print ("\\_public \\pdfname \\pdfstring ;")
```

2.39.5 Management of PDF page resources

Traditionally, pdfTeX allowed managing PDF page resources (graphics states, patterns, shadings, etc.) using a single toks register, `\pdfpageresources`. This is insufficient due to the expected PDF object structure and also because many “packages” want to add page resources and thus fight for the access to that register. We add a finer alternative, which allows adding different kinds of resources to a global page resources dictionary. Note that some resource types (fonts and XObjects) are already managed by LuaTeX and shouldn’t be added!

XObject forms can also use resources, but there are several ways to make LuaTeX reference resources from forms. It is hence left up to the user to insert page resources managed by us, if they need them. For that, use `pdf.get_page_resources()`, or the below TeX alternative for that.

```
local resource_dict_objects = {}
local page_resources = {}
function pdf.add_page_resource(type, name, value)
    local resources = page_resources[type]
    if not resources then
        local obj = pdf.reserveobj()
        pdf.setpageresources(fmt("%s %s %s", pdf.get_page_resources(), pdf_name(type), pdf_ref(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, "_pageresources")
```

2.39.6 Handling of colors and transparency using attributes

Because LuaTeX doesn't do anything with attributes, we have to add meaning to them. We do this by intercepting TeX just before it ships out a page and inject PDF literals according to attributes.

The attribute for coloring is allocated in `colors.opm`

```
local color_attribute = registernumber("_colorattr")
local transp_attribute = registernumber("_transpattr")
```

Now we define function which creates whatsit nodes with PDF literals. We do this by creating a base literal, which we then copy and customize.

```
local pdf_base_literal = direct.new("whatsit", "pdf_literal")
setfield(pdf_base_literal, "mode", 2) -- direct mode
local function pdfliteral(str)
    local literal = copy(pdf_base_literal)
    setfield(literal, "data", str)
    return literal
end
optex.directpdfliteral = pdfliteral
```

The function `colorize(head, current, current_stroke, current_tr)` goes through a node list and injects PDF literals according to attributes. Its arguments are the head of the list to be colored and the current color for fills and strokes and the current transparency attribute. It is a recursive function – nested horizontal and vertical lists are handled in the same way. Only the attributes of "content" nodes (glyphs, rules, etc.) matter. Users drawing with PDF literals have to set color themselves.

Whatsit node with color setting PDF literal is injected only when a different color or transparency is needed. Our injection does not care about boxing levels, but this isn't a problem, since PDF 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, and instead conservatively consider glyphs to always need stroke color. But users can set `\directlua{optex.glyphstroked=false}` if they are sure that their document doesn't include colorized fonts in mode 1 or 2. Then the PDF output can be smaller in its size (when there is huge amount of color switching, like in this document).

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.

```
optex.glyphstroked = true  -- default: insert stroke color for glyphs too.

local function is_color_needed(head, n, id, subtype) -- returns fill, stroke color needed
    if id == glyph_id then
        return true, optex.glyphstroked
    elseif id == glue_id then
        n = getleader(n)
        if n then
            return true, true
        end
    elseif id == rule_id then
        local width, height, depth = getwhd(n)
        if width <= one_bp or height + depth <= one_bp then
            -- running (-2^30) may need both
            return true, true
        end
        return true, false
    elseif id == whatsit_id and (subtype == pdfliteral_id
                or subtype == pdfsave_id
                or subtype == pdfrestore_id) then
        return true, true
    end
    return false, false
end

local function colorize(head, current, current_stroke, current_tr)
    for n, id, subtype in traverse(head) do
        if id == hlist_id or id == vlist_id then
            -- nested list, just recurse
            local list = getlist(n)
            list, current, current_stroke, current_tr =
               colorize(list, current, current_stroke, current_tr)
            setlist(n, list)
        elseif id == disc_id then
            -- at this point only no-break (replace) list is of any interest
            local replace = getfield(n, "replace")
            if replace then
                replace, current, current_stroke, current_tr =
                    colorize(replace, current, current_stroke, current_tr)
                setfield(n, "replace", replace)
            end
        else
            local fill_needed, stroke_needed = is_color_needed(head, n, id, subtype)
            local new = getattribute(n, color_attribute) or 0
            local newtr = getattribute(n, transp_attribute) or 0
            local newliteral = nil
            if current ~= new and fill_needed then
                newliteral = token_getmacro("_color:"..new)
                current = new
            end
            if current_stroke ~= new and stroke_needed then
                local stroke_color = token_getmacro("_color-s:"..current)
                if stroke_color then
                    if newliteral then
                        newliteral = fmt("%s %s", newliteral, stroke_color)
                    else
                        newliteral = stroke_color
                    end
                    current_stroke = new
                end
            end
            if newtr ~= current_tr and fill_needed then -- (fill_ or stroke_needed) = fill_neded
                if newliteral ~= nil then
                    newliteral = fmt("%s /tr%d gs", newliteral, newtr)
                else
                    newliteral = fmt("/tr%d gs", newtr)
                end
                current_tr = newtr
            end
            if newliteral then
                head = insertbefore(head, n, pdfliteral(newliteral))
            end
        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 whatsits) 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 color_count = registernumber("_colorcnt")
local function set_node_color(n, color) -- "1 0 0 rg" or "0 g", etc.
    local attr = tonumber(token_getmacro("_color::"..color))
    if not attr then
        attr = tex_getcount(color_count)
        tex_setcount(color_count, attr + 1)
        local strattr = tostring(attr)
        token_setmacro("_color::"..color, strattr, "global")
        token_setmacro("_color:"..strattr, color, "global")
        token_setmacro("_color-s:"..strattr, string.upper(color), "global")
    end
    setattribute(todirect(n), color_attribute, attr)
end
optex.set_node_color = set_node_color
```

```
function optex.hook_into_luaotfload()
    -- color support for luaotfload v3.13+, otherwise broken
    pcall(luaotfload.set_colorhandler, function(head, n, rgbcolor) -- rgbcolor = "1 0 0 rg"
        set_node_color(n, rgbcolor)
        return head, n
    end)

    -- transparency support for luaotfload v3.22+, otherwise broken
    pcall(function()
        luatexbase.add_to_callback("luaotfload.parse_transparent", function(input) -- from "00" to "FF"
            -- in luaotfload: 0 = transparent, 255 = opaque
            -- in optex:      0 = opaque,      255 = transparent
            local alpha = tonumber(input, 16)
            if not alpha then
                tex.error("Invalid transparency specification passed to font")
                return nil
            elseif alpha == 255 then
                return nil -- this allows luaotfload to skip calling us for opaque style
            end
            local transp = 255 - alpha
            local transpv = fmt("%.3f", alpha / 255)
            pdf.add_page_resource("ExtGState", fmt("tr%d", transp), pdf_dict{ca = transpv, CA = transpv})
            pdf.add_page_resource("ExtGState", "tr0", pdf_dict{ca = 1, CA = 1})
            return transp -- will be passed to the below function
        end, "optex")

        luaotfload.set_transparenthandler(function(head, n, transp)
            setattribute(n, transp_attribute, transp)
            return head, n
        end)
    end)
end

```

`\_beglocalcontrol` ⟨*tokens*⟩ `\_endlocalcontrol` runs ⟨*tokens*⟩ fully at expand processor level despite the fact that ⟨*tokens*⟩ processes unexpandable commands.

```

define_lua_command("_beglocalcontrol", function()
   return tex.runtoks(token.get_next)
end)

   -- History:
   -- 2025-05-12 optex.glyphstoked introduced
   -- 2025-05-11 if not \vbox then raw_ht retunts zero instead error
   -- 2024-12-18 \pdfstring etc. introduced
   -- 2024-09-06 raw_ht() implemented
   -- 2024-06-02 more checking in add_to_callback and remove_from_callback
   -- 2024-02-18 \_beglocalcontrol added
   -- 2022-08-25 expose some useful functions in `optex` namespace
   -- 2022-08-24 luaotfload transparency with attributes added
   -- 2022-03-07 transparency in the colorize() function, current_tr added
   -- 2022-03-05 resources management added
   -- 2021-07-16 support for colors via attributes added
   -- 2020-11-11 optex.lua released
```

2.40 Printing documentation

The `\printdoc` ⟨*filename*⟩⟨*space*⟩ and `\printdoctail` ⟨*filename*⟩⟨*space*⟩ commands are defined after the file `doc.opm` is load by `\load [doc]`.

The `\printdoc` starts reading of given ⟨*filename*⟩ from the second line. The file is read in *the listing mode*. The `\prindoctail` starts reading given ⟨*filename*⟩ from the first occurrence of the `\_endcode`. The file is read in normal mode (like `\input` ⟨*filename*⟩).

The *listing mode* prints the lines as a listing of a code. This mode is finished when first `␣␣\_doc` occurs or first `\_endcode` occurs. At least two spaces or one tab character must precede before such `\_doc`. On the other hand, the `\_endcode` must be at the left edge of the line without spaces. If this rule is not met then the listing mode continues.

If the first line or the last line of the listing mode is empty then such lines are not printed. The maximal number of printed lines in the listing mode is `\maxlines`. It is set to almost infinity (100000). You can set it to a more sensible value. Such a setting is valid only for the first following listing mode.

When the listing mode is finished by `\_doc` then the next lines are read in the normal way, but the material between `\begtt` ... `\endtt` pair is shifted by three letters left. The reason is that the three spaces of indentation is recommended in the `\_doc` ... `\_cod` pair and this shifting is compensation for this indentation.

The `\_cod` macro ignores the rest of the current line and starts the listing mode again.

When the listing mode is finished by the `\_endcode` then the `\endinput` is applied, the reading of the file opened by `\printdoc` is finished.

You cannot reach the end of the file (without `\_endcode`) in the listing mode.

The main documentation point can document a control sequence (use `` \`\⟨sequence⟩` ``) or a normal sequence without backslash (use `` \`⟨text⟩` ``). It is printed in red. Examples: `` \`\foo` `` (control sequence), `` \`foo` `` (normal sequence). The user documentation point is the first occurrence of `` \^`\⟨sequence⟩` `` or `` \^`⟨text⟩` `` There can be more such markups, all of them are hyperlinks to the relevant main documentation point. And main documentation point is a hyperlink to the relevant user documentation point if this point precedes. Finally, the `` \~`\⟨sequence⟩` `` (for example `` \~`\foo` ``) are hyperlinks to the user documentation point.

By default, the hyperink from main documentation point to the user documentation point is active only if it is backward link, i.e. the main documentation point is given later. The reason is that we don't know if such user documentation point will exist when creating main documentation point and we don't want broken links. If you are sure that user documentation point will follow then use prefix `\fw` before `` \` ``, for example `` \fw\`\foo` `` is main documentation point where the user documentation point is given later and forward hyperlink is created here.

Documented 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. Moreower, active links are control sequences of the type `\_foo` or `\.foo` although the documentation mentions only `\foo`. Another text is printed in black.

The listing mode is able to generate external links to another OpTeX-like documentation, if the macro `\el:`⟨*csname*⟩ is defined. The macro should create a hyperlink using `\_tmpa` where the link name of the ⟨*csname*⟩ is saved and `\_tmpb` where the name of the ⟨*csname*⟩ to be printed is saved (`\tmpb` is without backlash but it can include preceding `_` or `.` unlike `\_tmpa`). For example, suppose, that we have created `optex-doc.eref` file by:

```
TEXINPUTS='.;$TEXMF/{doc,tex}//' optex optex-doc
grep Xindex optex-doc.ref > optex-doc.eref
```

The `.eref` file includes only `\_Xindex{;`⟨*csname*⟩`}{}` lines from `optex-doc.ref` file. Then we can use following macros:

```
\def\_Xindex#1#2{\slet{el:\ignoreit#1}{optexdoclink}}
\def\optexdoclink{%
   \edef\extlink{\optexdocurl\csstring\#cs:\_tmpa}%
   \xlink{url}{\extlink}{\Cyan}{\csstring\\\_tmpb}}
\def\optexdocurl{http://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf}
\isfile{optex-doc.eref}\iftrue \input{optex-doc.eref}\fi
```

All `\el:`⟨*csname*⟩, where ⟨*csname*⟩ is from `optex-doc.ref`, have the same meaning: `\optexdoclink` in this example. And `\optexdoclink` creates the external link in `\Cyan` color.

2.40.1 Implementation

doc.opm

```
\_codedecl \printdoc {Macros for documentation printing <2025-05-15>} % loaded on demand by \load[doc]
```

General declarations.

doc.opm

```
\_fontfam[lmfonts]

\_let \mlinkcolor=\Red    % main doc. points
\_let \ulinkcolor=\Blue   % user doc. points
\_let \fnamecolor=\Brown  % file names in listing headers
\_def \bgverbcolor  {\_setcmykcolor{0 0 .3 .03}} % background for listings
\_def \outlinkcolor {\_setcmykcolor{1 0 1 .2}}   % green for outerlinks
\_def \inlinkcolor  {\_setcmykcolor{0 1 0 .1}}   % magenta for internal links
\_hyperlinks \inlinkcolor \outlinkcolor
\_enlang
\_enquotes
```

Maybe, somebody needs `\seccc` or `\secccc`?

doc.opm

```
\_eoldef\seccc#1{\_medskip \_noindent{\_bf#1}\_par\_nobreak\_firstnoindent}
\_def\secccc{\_medskip\_noindent $\_bullet$ }
```

`\enddocument` can be redefined.

doc.opm

```
\_let\enddocument=\_bye
```

A full page of listing causes underfull `\vbox` in output routine. We need to add a small tolerance.

doc.opm

```
\_pgbottomskip=0pt plus10pt minus2pt
```

The listing mode is implemented here. The `\maxlines` is maximal lines of code printed in the listing mode.

doc.opm

```
\_newcount \_maxlines   \_maxlines=100000
\_public \maxlines ;

\_eoldef\_cod#1{\_par \_wipeepar
   \_vskip\_parskip \_medskip \_ttskip
   \_begingroup
   \_typosize[8/10]
   \_let\_printverbline=\_printcodeline
   \_ttline=\_inputlineno
   \_setverb
   \_ifnum\_ttline<0 \_let\_printverblinenum=\_relax \_else \_initverblinenum \_fi
   \_adef{ }{}\_adef^I{\t}\_parindent=\_ttindent \_parskip=0pt
   \_def\t{\_hskip \_dimexpr\_tabspaces em/2\_relax}%
   \_relax \_ttfont
   \_endlinechar=`^^J
   \_def\_tmpb{\_start}%
   \_readverbline
}
\_def\_readverbline #1^^J{%
   \_def\_tmpa{\_empty#1}%
   \_let\_next=\_readverbline
   \_ea\_isinlist\_ea\_tmpa\_ea{\_Doc}\_iftrue \_let\_next=\_processinput \_fi
   \_ea\_isinlist\_ea\_tmpa\_ea{\_Doctab}\_iftrue \_let\_next=\_processinput \_fi
   \_ea\_isinlist\_ea\_tmpa\_ea{\_Endcode}\_iftrue \_def\_next{\_processinput\_endinput}\_fi
   \_ifx\_next\_readverbline \_addto\_tmpb{#1^^J}\_fi
   \_next
}
{\_catcode`=13 \_gdef\_aspace{ }}\_def\_asp{\_ea\_noexpand\_aspace}
\_edef\_Doc{\_asp\_asp\_bslash _doc}
\_bgroup \_lccode`~=`^I \_lowercase{\_egroup\_edef\_Doctab{\_noexpand~\_bslash _doc}}
\_edef\_Endcode{\_noexpand\_empty\_bslash _endcode}
```

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.

doc.opm

```
\_def\_makecs{\_def\_tmpa{}\_futurelet\_next\_makecsD}
\_def\_makecsD{\_if.\_next \_ea\_makecsB \_else \_ea\_makecsA \_fi} % \.foo is accepted
\_def\_makecsA{\_ifcat a\_noexpand\_next \_ea\_makecsB \_else \_ea\_makecsF \_fi}
\_def\_makecsB#1{\_addto\_tmpa{#1}\_futurelet\_next\_makecsA}
\_def\_makecsF{\_let\_tmpb=\_tmpa
   \_ifcsname el:\_tmpa\_endcsname \_csname el:\_tmpa\_endcsname \_returnfi \_fi
   \_ifcsname ,;\_tmpa\_endcsname \_intlink \_returnfi \_fi
   \_remfirstunderscoreordot\_tmpa
   \_ifcsname el:\_tmpa\_endcsname \_csname el:\_tmpa\_endcsname \_returnfi \_fi
   \_ifcsname ,;\_tmpa\_endcsname \_intlink \_returnfi \_fi
   \_csstring\_tmpb \_relax
}
\_def\_processinput{%
   \_let\_start=\_relax
   \_ea\_replstring\_ea\_tmpb\_ea{\_aspace^^J}{^^J}
   \_addto\_tmpb{\_fin}%
   \_isinlist\_tmpb{\_start^^J}\_iftrue \_advance\_ttline by1\_fi
   \_replstring\_tmpb{\_start^^J}{\_start}%
   \_replstring\_tmpb{\_start}{}%
   \_replstring\_tmpb{^^J\_fin}{\_fin}%
   \_replstring\_tmpb{^^J\_fin}{}%
   \_replstring\_tmpb{\_fin}{}%
   \_ea\_prepareverbdata\_ea\_tmpb\_ea{\_tmpb^^J}%
   \_replthis{\_csstring}{\_noexpand\_makecs}%
   \_ea\_printverb \_tmpb\_fin
   \_par
   \_endgroup \_ttskip
   \_isnextchar\_par{}{\_noindent}%
}
\_def\_remfirstunderscoreordot#1{\_ea\_remfirstuordotA#1\_relax#1}
\_def\_remfirstuordotA#1#2\_relax#3{\_if _#1\_def#3{#2}\_fi \_if\_string#1.\_def#3{#2}\_fi}
```

By default the internal link is created by `\_intlink` inside listing mode. But you can define `\el:`⟨*csname*⟩ which has precedence and it can create an external link. The `\_tmpa` includes the name used in the link and `\_tmpb` is the name to be printed. See `\_makecsF` above and the example at the beginning of this section.

doc.opm

```
\_def\_intlink{\_link[cs:\_tmpa]{\ulinkcolor}{\_csstring\_tmpb}}
```

The lines in the listing mode have a yellow background.

doc.opm

```
\_def\_printcodeline#1{\_advance \_maxlines by-1
   \_ifnum \_maxlines<0 \_ea \_endverbprinting \_fi
   \_ifx\_printfilename\_relax \_penalty \_ttpenalty \_fi \_vskip-4pt
   \_noindent\_rlap{\bgverbcolor \_vrule height8pt depth5pt width\_hsize}%
   \_printfilename
   \_indent \_printverblinenum #1\_par}

\_def\_printfilename{\_hbox to0pt{%
   \_hskip\_hsize\_vbox to0pt{\_vss\_llap{\fnamecolor\docfile}\_kern7.5pt}\_hss}%
   \_let\_printfilename=\_relax
}
\_everytt={\_let\_printverblinenum=\_relax}

\_long\_def\_endverbprinting#1\_fin#2\_fin{\_fi\_fi \_global\_maxlines=100000
   \_noindent\_typosize[8/]\_dots etc. (see {\_tt\fnamecolor\docfile})}
```

`\docfile` is currently documented file.
`\printdoc` and `\printdoctail` macros are defined here.

doc.opm

```
\_def\docfile{}
\_def\_printdoc #1 {\_par \_def\docfile{#1}%
   \_everytt={\_ttshift=-15pt \_let\_printverblinenum=\_relax}%
   \_ea\_cod \_input #1
   \_everytt={\_let\_printverblinenum=\_relax}%
   \_def\docfile{}%
}
\_def\_printdoctail #1 {\_bgroup
   \_everytt={}\_ttline=-1 \_ea\_printdoctailA \_input #1 \_egroup}
{\_long\_gdef\_printdoctailA#1\_endcode{}}

\_public \printdoc \printdoctail ;
```

You can do `\verbinuput \vitt{`⟨*filename*⟩`} (`⟨*from*⟩`-`⟨*to*⟩`)` ⟨*filename*⟩ if you need analogical design like in listing mode.

doc.opm

```
\_def\_vitt#1{\_def\docfile{#1}\_ttline=-1
   \_everytt={\_typosize[8/10]\_let\_printverbline=\_printcodeline \_medskip}}

\_public \vitt ;
```

The Index entries of control sequences are with semicolumn instead backslash in `.ref` file. It is ignored when sorting but doesn't cause problems in the `.ref` file. The Index entries of normal sequences are finished by `|` because these entries should be sorted just after control sequences with the same name. When printing Index, we firts try to refer to the main documentation point (the `\cs:name` or `\ns:name` exists). Index entries with only user documentation points refers to this point. Other index entries are printed as usual without backslash.

doc.opm

```
\_addto \_ignoredcharsen {_}  % \foo, \_foo is the same in the fist pass of sorting
\_let\_optexprintii=\_printii
\_def\_printii #1#2&{%
   \_if ;#1\_def\_tmp{#2}\_else \_printiiB #1#2\_end|\_end\_relax{#1#2}\_fi % does \def\_tmp{#1#2}
   \_ea\_definefirstii \_tmp&%
   \_ifx\_firstii\_lastii\_else
      \_ea\_newiiletter\_ea{\_firstii}{\_tmp}\_let\_lastii=\_firstii\_fi
   \_let\_currii=\_tmp \_the\_everyii \_noindent
   \_hskip-\_iindent \_ignorespaces
   \_if ;#1\_printiiC c\_bslash{#1#2}\_else \_printiiC n\_empty{#1#2}\_fi
}
\_def\_printiiB#1|\_end#2\_relax#3{\_ifx^#2^\_def\_tmp{#3}\_else\_def\_tmp{#1}\_fi}
\_def\_printiiC #1#2#3{% #1: c or n, #2: \_bslash or \_empty, \_tmp: name, #3: (raw name)
   \_printiiD #1#2{#3}% maybe other index types
   \_ifcsname #1s:\_tmp\_endcsname {\_tt\_link[#1s:\_tmp]\ulinkcolor{#2\_tmp}}\_returnfi\_fi
   \_ifcsname #1s:^\_tmp\_endcsname {\_tt\_link[#1s:^\_tmp]\ulinkcolor{#2\_tmp}}\_returnfi\_fi
   \_isnextchar<{\_ea\_printiiA\_ignoreit}%
      {\_isnextchar>{\_ea\_printiiA\_ignoreit}{\_printiiA}}#3//% other index types
   \_relax \_space
}
\_def\_printiiD #1#2#3{} % macro programmer can redefine it and declare another index types
\_def\_pgprintA #1{#1}   % no hyperlinks from page numbers

\_def\_printiipages#1&{\_let\_pgtype=\_undefined \_tmpnum=0
   {\_rm\_printpages #1,:,\_par}}

\_sdef{_tocl:1}#1#2#3{\_nofirst\_bigskip
   \_bf\_llaptoclink{#1}{#2}\_hfill \_pgn{#3}\_tocpar\_medskip}
```

If this macro is loaded by `\load` then we need to initialize catcodes using the `\_afterload` macro.

doc.opm

```
\_def\_afterload{\_catcode`=13 \_catcode`=13
   \_wlog {doc.opm: catcodes of < and ` activated.}%
}
```

The `<something>` will be print as ⟨*something*⟩.

doc.opm

```
\_let\lt=<
\_catcode`=13

\_def<#1>{$\langle\hbox{\it#1}\rangle$}
\_everyintt{\_catcode`=13 \_catcode`\.=11 }
```

Main documentation point to a control sequence is created by `\`\foo`` and its link name is `cs:foo`. Main documentation point to another sequence (without backslash) is created by `\`foo`` and its link name is `ns:foo`. User documentation point to a control sequence is created by `\^`\foo`` and its link name is `cs:^foo`. User documentation point to another sequence (without backslash) is created by `\^`foo`` and its link name is `ns:^foo`. Only the first occurrence of the user documentation point has its link destination. User documentation point are linked to relevant main documentation points. When the link destination is created then the control sequence with the same name is defined as an empty macro. Users can create a link to the user documentation point by `\~`\foo`.

doc.opm

```
\_def\_docrefcodes{\_catcode`\.=11\_relax}

\_verbchar`

\_def{\_bgroup \_docrefcodes \_mainpoint}
\_def\_mainpoint #1`{\_egroup\_leavevmode\_edef\_tmp{\_csstring#1}%
   \_ea \_mainpointA \_string#1\_relax % \defines \_cn as c or n
   \_iindex{\_if\_cn c;\_tmp\_else\_tmp|\_fi}%
   \_ifcsname \_cn s:\_tmp\_endcsname \_moremainpoints \_else \_dest[\_cn s:\_tmp]\_fi
   \_sxdef{\_cn s:\_tmp}{}%
   \_hbox{\_ifcsname \_cn s:^\_tmp\_endcsname
            \_link[\_cn s:^\_tmp]{\mlinkcolor}{\_tt\_if\_cn c\_bslash\_fi \_tmp}\_else
            {\_tt \mlinkcolor \_if\_cn c\_bslash\_fi \_tmp}\_fi}%
}
\_def\_mainpointA #1#2\_relax {\_edef\_cn{\_if\_csstring#1c\_else n\_fi}}
\_def\_moremainpoints{\_opwarning{Second main documentation point \_if\_cn c\_bslash\_fi\_tmp}}

\_def`{\_bgroup \_docrefcodes \_docpoint}
\_def\_docpoint #1{\_egroup\_leavevmode\_edef\_tmp{\_csstring#1}%
   \_ea \_mainpointA \_string#1\_relax % \defines \_cn as c or n
   \_if\_cn c\_iindex{;\_tmp}%
      \_hbox{\_ifcsname cs:^\_tmp\_endcsname
             \_else \_dest[cs:^\_tmp]\_sxdef{cs:^\_tmp}{}\_fi
             \_link[cs:\_tmp]{\ulinkcolor}{\_tt\_string#1}}%
      \_afterfi{\_futurelet\_next\_cslinkA}%
   \_else \_afterfi {\_docpointN #1}\_fi
}
\_def\_cslinkA{\_ifx\_next`\_ea\_ignoreit \_else \_ea\_ea\_ea`\_ea\_string\_fi}
\_def\_docpointN #1`{\_iindex{#1|}%
      \_hbox{\_ifcsname ns:^#1\_endcsname \_else \_dest[ns:^#1]\_sxdef{ns:^#1}{}\_fi
             \_link[ns:#1]{\ulinkcolor}{\_tt#1}}%
}
\_def`{\_bgroup \_docrefcodes \_doctpoint}
\_def\_doctpoint #1{\_egroup\_leavevmode\_edef\_tmp{\_csstring#1}%\_iindex{\_tmp}%
   \_ea \_mainpointA \_string#1\_relax % \defines \_cn as c or n
   \_if\_cn c\_iindex{;\_tmp}%
       \_hbox{\_link[cs:^\_tmp]{\ulinkcolor}{\_tt\_string#1}}%
       \_afterfi{\_futurelet\_next\_cslinkA}%
   \_else \_aferfi{\_docpointNA #1}\_fi
}
\_def\_docpointNA #1`{\_iindex{#1|}\_hbox{\_link[ns:^#1]{\ulinkcolor}{\_tt#1}}}
```

The `\fw` macro for forward links to user documentation point (given later) is defined here.

doc.opm

```
\_def\_fw#1`{{\_slet{cs:^\_csstring#1}{_fw}\_slet{ns:^\_csstring#1}{_fw}#1`}}
\_public \fw ;
```

Index

Control sequences declared by OpTEX have page list here and they are internal links to their main documentation point. TEX primitives used by OpTEX have no page list here and they are external links to TEX in a Nutshell to the place where the primitive is briefly described.