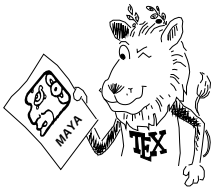
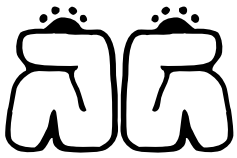
MayaPS: Typing Maya with TeX/LaTeX
Reference Manual. Version 1.0 (December 3, 2012)

Stepan Orevkov



I have written the system MayaPS in collaboration with Bruno Delprat (he formulated the principles of ancient Maya typesetting and created the Maya fonts used here). The idea to use PostScript language for drawing composed glyphs belongs to Ilya Zakharevich.

Table of Contents



1. What is MayaPS for

The package MayaPS is designed for doing the paleography of ancient Maya texts using TEX or LATEX **and** Dvips. This text is typeset in LATEX using this package and you see the result. For example, to get the above glyph, I typed

"`\centerline{\mayaSize{2cm}\mayaGlyph{422.422}}`"

and to get  , I just typed

"`and to get \maya{(023.153.023):220}, I just typed`".

Note, that this is a really existing glyph.

Ancient Maya glyphs are composed by attaching together primitive (indecomposable) glyphs. Some specialists think that they correspond to syllables, some other argue with them. MayaPS does not care of any grammatical or linguistical meaning of primitive glyphs. They are just graphical images which are elementary bricks of Maya typesetting, like letters for European languages. Each complete glyph (composed of primitive ones) is rescaled so that it fills a rectangle of a fixed size called *cartouche* in this document. The cartouches are placed in a regular way on a page.

The glyphs  (non-existing) and  that we used above, are composed of primitive glyphs  (by the way, to get this, I typed "`...glyphs \maya{422 001 023 153 220}`").

A more interesting example – page 7b of the Dresden Codex:

(the paleography is due to Bruno Delprat). To obtain it, I typed

```
\noindent\mayaC{     % \mayaC = glyphs with captions
451.452  026.314/(314)  111.274         047.276/010  913  810
451.452  026.314/(314)  570/014.267.024 111.+176/111 913  810
451.452  026.314/(314)  245.234         026.172/023  913  810}
```

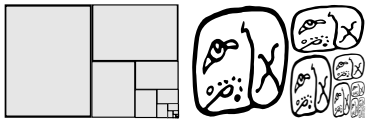
As it should be clear already, each primitive glyph is referred to by its name (called further the *glyph name*). So, the glyph names used above are "`422`", "`001`", "`023`", "`153`", and "`220`". In general, a glyph name is any sequence of digits `0...9` and letters `a...z`, `A...Z`. The encoding system is rather flexible. For example, after the command `\mayaDefine{A9z}{442}` you can type "`\maya{A9z}`" to get  . MayaPS supports several Maya fonts. For example, the beginning of the above citation from the Dresden Codex printed in the font

'gates' looks as

451.452 026.314/(314) 111.274 047.276/010 913 etc.

(I typed \gates and then, just copied the above codes). This is a font designed (and made in plumb!) by William Gates in the 30's and adapted for MayaPS by Bruno Delprat. In this document we use mostly the font 'codex' created by Bruno Delprat with use of the tools described in §12.

If  and are two glyphs (primitive or not), then A.B and A:B encode the glyphs and . To control the order of composition, one can use parentheses in the same way as in mathematical formulas. For example, the both A.B:C and A.(B:C) stand for but (A.B):C stands for (in glyph codes, '/' means the same as ':'). Any formula of this kind is admitted, even something like this



The picture on the right hand side (I don't say "glyph" because there are no such glyphs in ancient Maya language) is printed by the command

```
\maya{322.322:(322.322:(322.322:(322.322:(322.322:(322.322)))))}
```

Let us discuss again the glyph  . We see that the primitive glyph 023 occurs here in two different ways: and . Moreover, in  it occurs twice like this: . MayaPS automatically chooses the orientation of each primitive glyph according to "grammar rules" formulated by Bruno Delprat after a careful analysis of ancient manuscripts. Of course, these rules may have exceptions. It is very easy to handle them. For example, if you type \maya{422.001}, you obtain  (the default orientation), but if you type \maya{|422.001}, you obtain . The rules (and possibilities to avoid them) are described in §3. Different colors may be used (see §6).

MayaPS provides a tool to add or replace glyphs in existing fonts (this is very easy). In §11 and §12 we explain also how to create a new font.

MayaPS produces a PostScript (ps) file whose length is optimized. Suppose, you have already a ps file (produced by TEX/Dvips) in a European language, and you include ancient Maya glyphs into it. Then MayaPS adds to ps only:

- MayaPS header (8 Kb);
- definitions of primitive glyphs (0.5 – 3 Kb per glyph for 'codex');

- about 60 bytes for each composed glyph.

MayaPS includes the definitions of only those primitive glyphs which are really used in the text. Each definition is included only once even if the primitive glyph is used many times.

PostScript files can be printed out, converted to pdf, or viewed on the screen using, e.g., `ghostview`, `gv`, `ps2pdf` (Unix/Linux-X11), GSview (Windows), etc. The font '`codex`' is designed so that the resulting `pdf` file becomes even shorter than `ps`.

Maximal portability is one of main principles of MayaPS. Even if nobody continues supporting it (which may happen), it will work on all future platforms as long as TEX, Dvips, and PostScript are supported.

Another attempt to adapt TEX/LATEX for Native-American languages (including Olmec) was done in [7]. Our approach is very different from that.

2. Installation and running

MayaPS is available at `http://picard.ups-tlse.fr/~orevkov/mayaps.html`

2.1. How to work without any installation. Just copy the files `mayaps.tex`, `mayaps.pro`, `codex.mpf`, `red89.tex` (if needed), and `mpfmap.tex` (if needed) to the current directory (folder) and work. By *current directory* we mean the directory containing the `tex` file you are writing. If you use MayaPS fonts other than `codex.mpf` or if you add glyphs using `\mayaAddGlyph`, then place the corresponding `mpf` and/or `eps` files to the current directory also.

Be sure that the Dvips program is installed on your computer. Otherwise install it using the documentation for your TEX installation (of course, TEX also should be installed).

To use MayaPS macros, place the line

```
\input mayaps
```

somewhere near the beginning of your `tex` file If you use LATEX, place it somewhere between `\documentclass` and `\begin{document}` commands.

When you have prepared a `tex` file (say, `foo.tex`), compile it by the command '`tex foo`' issued from the command line (or '`latex foo`' if you use LATEX). You will obtain a file `foo.dvi` in the current directory. Then type the command '`dvips foo`' (or, maybe, '`dvips foo -o`', because sometimes `dvips` without '`-o`' sends the output to a printer) and you will obtain a file `foo.ps`. This is a PostScript file which can be viewed, printed, or converted to PDF.

If your TEX is integrated into a graphic interface environment, you should tune it so that it calls TEX or LATEX and Dvips.

2.2. Installation. If you use MayaPS in several directories (folders), it is convenient to put all MayaPS files discussed in §2.1 in a fixed place. It should be a directory recognized as 'TeX source' by your TEX installation. See [8] or ask your system administrator how to choose it. Usually, it is a new subdirectory

(say, `mayaps`, but you may use any name) of *root*`/tex/generic` where *root* is either `texmf` or a directory whose name contains '`texmf`'. Sometimes you need explicitely make TEX 'see' this directory. For example, in MiKTeX (Windows), you run 'MiKTeX Options' in Start menu, click 'Roots', select your *root* directory, and click 'Refresh FNDB'; in typical UNIX/Linux installation you run the command '`texhash` *root*', for example, '`texhash /usr/share/texmf`'.

2.3. Don't use PdfTEX with MayaPS. Use TEX/Dvips.

TEX, Dvips, and PdfTEX. TEX is a program written by Donald Knuth (see [4]). It reads a source file (which usually has the extension `.tex`) and produces a file in the format Dvi (DeVice Independent; file extension `.dvi`). The Dvi file describes how the output document must look like. To print out the document on a specific printer (or to see it on a screen), one needs a Dvi driver. In the 80th and 90th, numerous Dvi drivers were developed for different printers, display viewers etc.

Dvips is a program written by Tomas Rokicki, which converts a Dvi file into a PS file (a file in Adobe's PostScript Language). In 90th PostScript Language became de facto an international standard of printer interface. Dvips also became the most popular Dvi driver at that time. Besides the conversion `dvi` → `ps`, Dvips provides some tools to include PostScript graphics into a document prepared with TEX. These tools are used in MayaPS.

Since the turn of the century PostScript is being replaced by PDF (new Adobe's format). It is better adapted for modern realities. For example, you can see PDF files directly from Internet browsers, it is the basic graphical format for Mac, etc. Today everybody knows what is PDF, but only some TEX users still remember what is PostScript.

Of course, a three step conversion `tex` → `dvi` → `ps` → `pdf` is always possible, but Han The Thanh made a shortcut. He has written PdfTEX – a program which produces a PDF file directly from `tex` source. Many modern TEX installations call PdfTEX instead of TEX by default.

PdfTEX is 99% compatible with TEX but, unfortunately, MayaPS falls into the remaining 1%, because it is heavily based on the interface of Dvips. The reason is very serious: all algorithms of assembling a composed glyph are implemented on the PostScript programming language.

LATEX is an extension of TEX written by Leslie Lamport (see [5]) and further developed by the LATEX3 project team. Any modern installation of TEX includes it. MayaPS is compatible with most of LATEX features (this text is prepared with LATEX). Some known incompatibilities and ways to avoid them are discussed in §7.7.

Can one write MayaPDF? Yes, but I won't. By the following reasons.

1). I don't believe that MayaPS will ever have more than 100 users (even this is a very optimistic estimate).

2). The interaction between TEX and Dvips is used not only in MayaPS. There are no principal obstacles to incorporate Dvips' interface into PdfTEX and I hope that sooner or later somebody will do it.

3. Glyph codes and glyph orientations

3.1. Glyph codes. A first idea what is a glyph code, is given in §1. It is an expression like `A.B:C` or `B:(A.C:D)` where `A`, `B`,... are glyph names (i.e., names of primitive glyphs), maybe preceded by modifiers ('`|`', for example) which allow to change the orientation. Glyph codes (and parts of them) may be supplied by color indicators (see §6). Let us give formal definitions.

Glyph Name (called also *primitive glyph code*) is a sequence of digits `0...9` and letters `a...z`, `A...Z`. Glyph names are defined in font files. They can also be defined by `\mayaAddGlyph` or `\mayaImport` command (see §4.2). All the glyph names available in the fonts `codex` and `gates` can be found in the documents `codex-map.pdf` and `gates-map.pdf` (see §4.2 for more details).

Modifier Character is one of the characters: `| ' - + ? = * A a C c R r`

Modifier is any sequence of modifier characters and brackets `[ ]`. Brackets must be balanced. Modifier characters which are letters (i.e., `A`, `a`, `C`, `c`, `R`, or `r`) should be enclosed in brackets. Example: `*`, `[r[]]`, `|[a][R+]` are modifiers but `*R[+]` is not, because `R` is not in the brackets.

Glyph Code is either a glyph name or one of the expressions `(`A`)`, mA, A`.`B, A`:`B, A`/`B, where A and B are glyph codes (not necessarily primitive) and m is a modifier Example: `|(1A.[r]123):xyz` is a glyph code.

The meaning of the operations '`.`' and '`:`' is already explained in Section 2. The symbol '`/`' means always the same as '`:`'. The meaning of the modifiers will be explained in §3.4.

3.2. Glyph types. From MayaPS' point of view, there are two types of primitive glyphs: *central elements* and *affixes*. Affixes are further subdivided into *numerals* and *non-numeral affixes*.

Usually, the both dimensions (width and height) of central elements are close to each other: they look like a square slightly deformed. In contrary, one side of an affix is usually longer than the other. However, there are no formal restrictions on the proportions of primitive glyphs of any type.

For example, in the font '`codex`', the primitive glyphs  (`124`),  (`173`),  (`222`) are central elements,  (`023`),  (`063`) are non-numeral affixes, and  (`991`),  (`813`) are numerals.

The property to be an affix or a central element is attributed to each primitive glyph. This information is kept in the font file together with the natural orientation, natural proportions, and the graphical image of each glyph.

3.3. Glyph orientations. Here we describe the default rules for the choice of orientations of primitive glyphs. Here 'default rule' means a rule which is applied when a glyph code has no modifiers. These rules are proposed by Bruno Delprat after a careful analysis of ancient Maya manuscripts.

This subsection can be skipped on the first reading. Moreover, it is not necessary to read it at all for using MayaPS. If you want to type a composed

glyph, just type it as it is (without any modifiers), look at the result, and if you are not satisfied by the orientation of some primitive glyphs, correct it by modifiers as explained in §3.4.

Default rules for the orientations of primitive glyphs.

1). Any central element always appears in the same orientation. We call it *the natural orientation*.

2). Five standard orientations are attributed to each affix. We shall denote them L (left), U (up), R (right), D (down), and S (single). If `A` is an affix and `C` a central element, then `A` appears in these orientations in the glyphs `A.C`, `A:C`, `C.A`, `C:A`, `A` respectively. The S-orientation of an affix we shall also call *natural*. The standard orientations of each affix are defined in the font file. For the most of non-numeral affixes of the font '`codex`' these orientations are related to each other as in the following example (affix `504`):

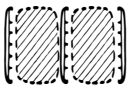
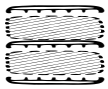
$L =$  $U =$  $R =$  $D =$  $S =$ 

However, there are some cases when this is not so, for example, affix `422`:

$L =$  $U =$  $R =$  $D =$  $S =$ 

3). Suppose that several glyphs are attached together horizontally. Suppose that there is at least one central element among them (composed glyphs are also considered here as central elements). Let C be the rightmost central element. Then all affixes which are to the left of C take the L-orientation and all affixes which are to the right of C take the R-orientation.

4). (Similar to the previous rule). Suppose that several glyphs are attached together vertically. Suppose that there is at least one central element among them (composed glyphs are also considered here as central elements). Let C be the lowermost central element. Then all affixes which are below C take the D-orientation and all affixes which are above C take the U-orientation.

Examples (non-existing):  `023.001.023.001.023`  `023:001:023:001:023`

5). (This rule has no analog for horizontal attachment). Suppose that only affixes are attached together vertically. Suppose also that one of the following cases takes place:

- 5.1). There are two consecutive identical non-numeral affixes.
- 5.2). There are two consecutive (not necessarily identical) numerals.

Then the two affixes (the uppermost pair of them if the choice is ambiguous) are places both in the D-orientation in Case 5.1 and in the U-orientation in Case 5.2. The combination of these two affixes is declared a central element and the rule 3 is applied.

Example (a really existing glyph): 

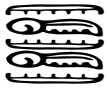
`023:023:415.130:176`

6). Suppose that only affixes are attached together horizontally. Then the leftmost one takes the L-orientation and all the others take the R-orientation.

7). Suppose that only affixes are attached together vertically. Suppose also that the rule 5) is not applicable. Then the uppermost affix takes the U-orientation and all the others take the D-orientation.

Example (non-existing):

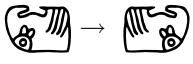
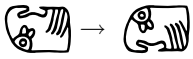
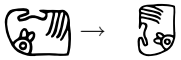
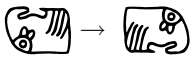
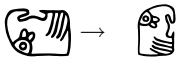
`023.023.023.023.023`



`023:504:023:504:023`

Note that `\maya{023.(023.023).023.023}` gives  because the composed glyph `(023.023)` is interpreted here as a central element by rule 3).

3.4. Modifiers. The modifiers `|`, `'`, `-`, `+`, `?`, `*`, `R`, and `r` change the orientation of a primitive glyph starting from the default orientation determined by the glyph's position:

`|` or `'`	Symmetry (the axis is '|')	
`-`	Symmetry (the axis is '—')	
`R` or `?`	Rotation by 90°	
`+`	Rotation by 180°	
`r` or `*`	Rotation by 270°	

('`+`' is chosen for 180°-rotation, because it is the same as '`|`' and then '`-`').
The modifiers `A`, `a`, `C`, `c`, and `=` switch the type of a primitive glyph:

`A` or `a` transform central elements to affixes;

`C`, `c`, or `=` transform affixes to central elements;

When several modifiers are applied to a glyph, they are applied one by one from the right to the left. For example, if you want to apply the symmetry '`|`' to the glyph `\maya{[R]314}` which is  , you add '`|`' like this `\maya{[|R]314}` and you obtain  .

4. Usage of different Maya fonts

4.1. Fonts. MayaPS supports several Maya fonts. At each moment of a `tex` file processing (since the command `\input mayaps` is executed) one of Maya fonts is current. It means that it is used as the *current Maya font* by the

commands (macros) of MayaPS. Before the first usage, each Maya font must be declared (loaded) by the command

`\mayaFont\`*cs*`=`*fontfile*

where `\`*cs* is any control sequence (backslash '`\`' followed by a chain of letters `a...z` or `A...Z`) and *fontfile* is the name without extension of a Maya font file. The file name must be followed by a space (end-of-line and tabulation are also treated by TEX as a space).

This command loads the font from the file *fontfile*`.mpf` and associates it to the control sequence `\`*cs*. The file *fontfile*`.mpf` must be 'visible' by TEX (for example, it can be placed in the current directory, see also §2.2). After this command, the loaded font can be made current by the command `\`*cs* more or less like for usual TEX fonts. Also similarly to usual TEX fonts, the action of `\`*cs* (which changes the current Maya font) is local, i.e. it acts till the end of the group (a *group* is, roughly speaking, a part of the `tex` file enclosed in braces `{ }`). See §7.6 for more details about local and global action of commands.

The font '`codex`' is already loaded and is current from the very beginning (i.e., after the command `\input mayaps`). It is associated to the control sequence `\codex`. To disable the automatic preloading of the font '`codex`', write `\def\mayaNoPreloadedFont{}` just before `\input mayaps`.

Example. Suppose that no font changes were done before (thus, the current font is '`codex`'). Then the commands

```
\mayaFont\th=thompson
\mayaFont\ga=gates
\maya{T520} \th \maya{T520} \maya{T520} \codex \maya{T520}
{ \ga \maya{T520} \maya{T520} } \maya{T520}
```

will produce the output: 

Compare this with the following example with usual TEX fonts:

```
Roman, \it Italic, \rm Roman, {\bf Bold Face,} again Roman
```

Roman, *Italic,* Roman, **Bold Face,** again Roman

4.2. Font map. Macro `\mpfmap`.

The macro `\mpfmap{`*font*`}` generates a list of all primitive glyphs of a font *font*`.mpf` (the font map) as well as a list of all substitutions (ligatures; see §5). This macro is defined in the file `mpfmap.tex`, thus, to make it available, use the command

```
\input mpfmap
```

(it is not necessary to write '`\input mayaps`' in this case because the file `mayaps.tex` is loaded from the file `mpfmap.tex` in the case when it was not loaded before).

The macro `\mpfmap` also creates two text files `map.tmp` (the font map) and `mapsubs.tmp` (the substitution table) which can be manually edited.

Example: Suppose that the file `codex-map.tex` contains only three lines

```
\input mpfmap
\mpfmap{codex}
\end
```

Then the command line

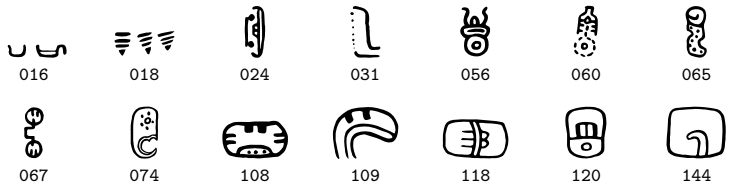
```
tex codex-map
```

creates three files: `codex-map.dvi`, `map.tmp`, and `mapsubs.tmp`. Further processing `codex-map.dvi` by Dvips, we obtain the file `codex-map.ps` which is

Map of Maya Font `codex.mpf` version 0.35 (Oct 27, 2012)

Primitive Glyphs

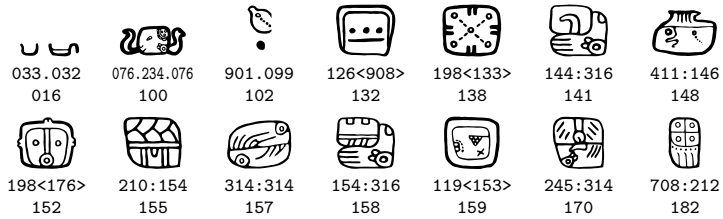
Irregular Affixes



. . . . . .

Substitutions (ligatures)

Ligature Glyphs



. . . . . .

The text file `map.tmp` is

```
% map.tmp - map of codex.mpf
% file generated by mpfmap v 1.0
\medskip \centerline {\bf Irregular Affixes}\nobreak \medskip
\mayaC {
016
018
...
065
}\par \noindent \mayaC {
067
 . . . . . .
```

and the text file `mapsubs.tmp` is

```
% mapsubs.tmp - substitution table of codex.mpf
% file generated by mpfmap v 1.0
\medskip \centerline {\bf Ligature Glyphs}\nobreak \medskip
\noindent
\mayaGlyphCC {033.032}\hs
\mayaGlyphCC {076.234.076}\hs
\mayaGlyphCC {901.099}\hs
 . . . . . .
```

4.3. Macros `\mayaAddGlyph` and `\mayaImport`.

4.3.1. Macro `\mayaAddGlyph[`$(L)(U)(R)(D)(S)$`]{`$name$`}`$type$`{`$file$`}`

defines a new glyph (or replaces an existing one) in the current font. The last argument must be followed by a space (or end-of-line, or tabulation).
Arguments:

- `[`$(L)(U)(R)(D)(S)$`]` (optional, i.e., may be omitted): the array of default orientations (only for affixes). Each of L, U, R, D, S is a string (maybe empty) of the modifiers `| ' - + * ? R r` (see §3.4). The default value (when the argument is omitted) is `[()(|r)(|)(R)()]`.
- $name$: the glyph name.
- $type$: the glyph type: one of the symbols `A, a` (both for affix), `C`, or `c` (both for central element).
- $file$ (optional): the name of an `eps` file with the glyph image. The default value is $name$`.eps` (note, that if this argument is used, the file name must be written with the extension; no extension is added automatically).

If a central element is defined by `\mayaAddGlyph`, then its natural orientation always coincides with that from the `eps` file.

Example 1. The command

```
\mayaAddGlyph{abc}c
```

creates (or replaces) a glyph in the current Maya font. Its name is `abc`, it is a central element, and its graphical image is loaded from the file `abc.eps`.

Example 2. The command

```
\mayaAddGlyph[(r)()(R)()()]{123}A{a.eps}
```

loads the affix `123` from the file `a.eps`. The U, D, and S orientations of this glyph coincide with the orientation from the `eps` file. The L orientation is obtained from it by turning the image from the file clockwise (respectively, counterclockwise for the R orientation).

Example 3. Assume that the file `empty.mpf` is the empty Maya font (see Example 1 in §10.5). Then the commands

```
\mayaFont\a=empty                \mayaFont\b=empty
\a\mayaAddGlyph{001}c{a.eps}     \b\mayaAddGlyph{001}c{b.eps}
```

load two Maya fonts `\a` and `\b` and then declare the glyph `001` in each of these fonts. The glyph `001` of the font `\a` is loaded from the file `a.eps` and the glyph `001` of the font `\b` is loaded from the file `b.eps`.

Restrictions on the EPS file. The file used in the `\mayaAddGlyph` must be an Encapsulated PostScript (`eps`) file, but not any `eps` file is allowed. It cannot contain '`currentfile`' command. Usually this command appears in `eps` files when bitmaps (rasterized images) are included there. I checked that `eps` files produced by `xfig` are good for `\mayaAddGlyph` if bitmaps are not inserted into them. Files produced by `autotrace` and `cotrace` are also good.

If colors are explicitely defined in the `eps` file, then the color mechanism described in §6 will not work for the corresponding glyph.

If you want to include a glyph which you have in a bitmap format (`bmp`, `tiff`, etc.) or a `jpeg` file, for example, if you scanned a hand-made picture, then you can vectorize it. The recommended vectorizer is `cotrace` which is supplied with the package MayaPS (see §12). The font '`codex`' used here is prepared with it. Any other vectorizer can be used as well.

The macro `\mayaAddGlyph` is retroactive. This means that if you use it to redefine an existing glyph name which was already used on earlier pages, then the new graphical image of the glyph (loaded from the `eps` file) will appear on the earlier pages also. If the glyph was used on the same page, but before `\mayaAddGlyph`, then it will not be redefined. This is why I recommend to put all the commands `\mayaAddGlyph` at the beginning of the file (but, of course, after `\input mayaps`).

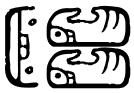
4.3.2. Macro `\mayaImport[`$(L)(U)(R)(D)(S)$`]{`g_{new}`}\`cs`{`g_{old}`}`$type$

is similar to `\mayaAddGlyph`. It (re)defines the glyph g_{new} of the current font to be equal to an existing glyph g_{old} of the font associated to $\backslash cs$. The font $\backslash cs$ may or may not coincide with the current font. The last argument must be followed by a space (or end-of-line, or tabulation).
Arguments:

- `[`$(L)(U)(R)(D)(S)$`]` (optional): same as for `\mayaAddGlyph`. The default value is inherited from g_{old}
- g_{new}: the glyph name to define in the current font.
- $\backslash cs$: a control sequence defined earlier by `\mayaFont\`cs`=`$fontfile$.
- g_{old}: the name of the glyph to be imported from the font $\backslash cs$.
- *type* (optional): the type (`A`, `a`, `C` or `c`). The type of g_{old} is used by default.

Example 1. Affixes (in MayaPS' sense) are not defined in the font `thompson`. However, they can be borrowed from the font `thompson2`:

```
\mayaFont\th=thompson  \mayaFont\thtwo=thompson2
\th \mayaImport[()(|r)(|)(R)()]{T1i}\thtwo{T1}
% \th \mayaImport{T1i}\thtwo{T1} gives the same result
Compare \th\mayaC{T1i.T671:T671}
with \codex\mayaC{026.314:(314)} and \gates\mayaC{026.314:(314)}
```

Compare  with  and 

`T1i.T671:T671` font '`thompson`' with imported glyphs

`026.314:(314)` font '`codex`'

`026.314:(314)` font '`gates`'

Remark 1. If you write `\codex\mayaImport{001}\codex{001}`, then you get a warning message from MayaPS and the command has no effect. However two-step and deeper 'self-importing' is not detected and it leads to a corrupted `ps` file. Be careful for not to do it.

Remark 2. The macro `\mayaImport` is retroactive (see the end of §4.3.1).

5. Ligatures and substitutions

5.1. Ligatures. Each Maya font may contain (and the font '`codex`' does contain) *ligatures*. Recall that a ligature in the Latin alphabet means that a publishing system (TEX, for example) prints **ffi** (a single glyph) instead of **ffi** when you type `ffi`.

A ligature table is defined in the font file. Each Maya font has its own ligature table. For example, in the font '`codex`', when you type `\maya{070/349}`, you get  instead of  . The ligature mechanism of MayaPS just replaces '`070/349`' by '`353`' in glyph codes before interpreting them. So, if you type directly `\maya{353}`, you obtain the same result  .

A complete list of all ligatures of any given font can be viewed as explained in §4.2.

If you want to get  instead of  , you may type `\maya{(070)/349}`. There is also a more permanent solution: the command

```
\mayaDeleteLigature{070/349}
```

cancels this ligature. Note that `\maya{{070}.349}` gives nonetheless  though `f{f}i` gives **ffi**.

A new ligature may be created by the command `\mayaAddLigature` which is just another name of the command `\mayaDefine` explained in §5.4.

The ligature `\maya{070/349}` discussed above has the following property. If we cancel it by the command `\mayaDeleteLigature{070/349}`, then the code `070/349` still makes sense. However, there are ligatures in the font '`codex`', which do not have this property, namely the ligatures containing the construction *A*<*B*> which means that the glyph `B` is inserted inside the glyph `A`. For

example, `\maya{369<023:023>}` produces  . Here the composed glyph `023:023` =  is inserted into `369` =  . The obtained glyph has its own name `373`, i.e., the commands `\maya{369<023:023>}` and `\maya{373}` are equivalent. Other examples are:

`173` =  inside `321` =  gives `321<173>` = `327` =  ;

`204` =  inside `369` =  gives `369<204>` = `367` =  .

These examples illustrate that it is very difficult to formulate a general rule for insertion of arbitrary glyphs into each other. Moreover, there are very few examples of such insertions. This is why we did not implement any general insertion mechanism in MayaPS. If a ligature `A<B>` is not defined, then `A<B>` is not a valid glyph code. For example,

```
\mayaDeleteLigature{321<173>} \maya{321<173>}
```

gives  which means that the glyph is not found in the current font.

5.2. Different encodings (catalogs) and phonetic notation

The glyph names in the font '`codex`' are given according to the catalog of Maya glyphs composed by Bruno Delprat on the base of the catalog from the book [3]. However, many specialists in ancient Maya are more familiar with Thompson's catalog [9]. The codes from Thompson's catalog are also included into the font '`codex`' using absolutely the same mechanism as is used for ligatures. For example, if you type `\maya{T1.001}`, you obtain  ('`T1`' is replaced with '`026`' before interpreting the glyph code). It is easy to define other encodings using the command `\mayaDefine` discussed in the next subsection.

Many glyphs in the font '`codex`' also may be referred to by their phonetic names and/or translations. For example, the font codex has a substitutions `u` → `026` (the phonetic value of the affix ), `death` → `047` and `muerte` → `047` (English and Spanish translations of ). So, typing any of `\maya{026}`, `\maya{T1}`, `\maya{u}` you obtain and typing any of `\maya{047}`, `\maya{T15}`, `\maya{death}`, `\maya{muerte}` you obtain .

Note, that from MayaPS' point of view there is no difference between ligatures, phonetic names, and different encodings. All of them are just substitution rules (see the next subsection) and they are treated by the same algorithms. A complete list of all substitution rules for any given font can be viewed as explained in §4.2.

5.3. Substitutions. Each Maya font may have *substitution rules*. Some of them may be predefined in the font file. For example, in '`codex`', the predefined substitution rules are ligatures, Thompson codes, and phonetic names which are discussed above. A complete list of all predefined substitution rules for any given font can be viewed as explained in §4.2. It can be seen also directly in the font file (the lines starting with `%L@`).

In this text we shall denote substitution rules by $s_1 \to s_2$ where s_1 and s_2 are two strings (chains of characters) which do not contain spaces. For example,

in the previous two subsections we discussed the following substitution rules in the font `codex`:

`070/349` → `353` `321<173>` → `327` `T1` → `026`
`369<023:023>` → `373` `369<204>` → `367`

All substitution rules of the current font are applied to an argument of the command `\maya` (and of other commands dealing with glyph codes) before passing the argument to the command. It is not necessary that the argument itself is a valid glyph code in the sense of §3.1 (for example, `321<173>` is not). It is important only that the result of substitutions is a valid glyph code.

Substitutions are applied only once (i.e., non-recursively). For example, `\maya{T64/349}` gives  , but not  . This means that the substitution `T64` → `070` is applied but the substitution (ligature) `070/349` → `353` is not applied to the result.

Substitutions are applied to a string by the following algorithm. First, we try to apply substitution rules to the longest possible initial substring. Then we find the first occurence of one of `] ( . : / ' | - + ? * =` in the rest of the string and try to apply substitution rules to the longest substring starting from the next character. And so on. For example, if the substitution rules

`xx.` → `1`, `.x` → `2`, `x.` → `3`, `xx` → `4`

are defined for a current font, then `\maya{xx.xx.xx.xx}` will produce the same result as `\maya{1xx.1xx}`.

5.4. Definition/canceling of substitutions rules.

A substitution $s_1 \to s_2$ for a current Maya font can be defined/canceled by the commands `\mayaDefine{`s_1`}{`s_2`}` and `\mayaUndefine{`s_1`}`.

`\mayaAddLigature{`s_1`}{`s_2`}` and `\mayaDeleteLigature{`s_1`}{`s_2`}` are equivalent versions of these commands.

Of course, all characters mentioned in §3.1 may be used in s_1 and in s_2. Also '`<`' and '`>`' are allowed in s_1. I do not recommend to use other characters. Even if you find experimentally that some of them give a reasonable result, this can be changed in future versions of MayaPS.

The action of the commands `\mayaDefine` and `\mayaUndefine` is local (see §7.6). For example, if you type

```
\maya{422} { \mayaDefine{422}{001} \maya{422} } \maya{422}
```

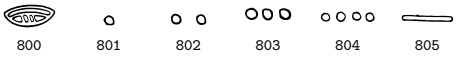
then you obtain  (the substitution acts only inside the braces). The global versions of these commands are:

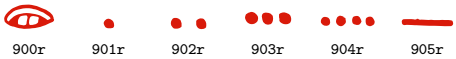
`\mayaGlobalDefine{`s_1`}{`s_2`}` `\mayaGlobalAddLigature{`s_1`}{`s_2`}`
`\mayaGlobalUndefine{`s_1`}` `\mayaGlobalDeleteLigature{`s_1`}`

5.5. Red numerals.

The red color of numerals has a semantic meaning in ancient Maya scripts. Red numerals can be typed in red as explained in §6. In monochrome texts, they can be represented by outlines:

Black numerals:  etc.
900 901 902 903 904 905

Red numerals (B/W):  etc.
800 801 802 803 804 805

Red numerals (colored):  etc.
900r 901r 902r 903r 904r 905r

The macro `\mayaRed` declares the substitutions `800` → `900r`, `801` → `901r`, etc. in the current font and the macro `\mayaBW` cancels them. Example:

```
\mayaRed\maya{803:805 001:803} \mayaBW\maya{803:805 001:803}
```

yields: 

The macro `\codexBW` is defined by

```
\def\codexBW{\mayaIgnoreRGB \codex \mayaBW}
```

It is convenient to use the macros `\mayaRed` and `\mayaBW` if you want to prepare a document which can be printed on a black-and-white printer, but shown in colors on a screen. In this case, you can use only the codes `800`, `801`, etc. for red numerals. If you want to get a colored version of the text, put the line `\mayaRed` just after `\input mayaps`. If you want to get a black-and-white version, replace it by `\mayaBW`.

The macros `\mayaRed` and `\mayaBW` make sense only for fonts containing black-and-white versions of red numerals. At the present time only the fonts '`codex`' and '`gates`' contain them. Moreover, these macros work properly only if the numerals are encoded in the same way as in the font '`codex`'. This is why the macros `\mayaRed` and `\mayaBW` are defined in a separate file red89.tex which is easy to adapt for future fonts based on other encodings. The file `red89.tex` is loaded however automatically from `mayaps.tex`.

5.6. Macro `\mayaDebug`. (see also `\mayaGlyphCC` in §7.5.1).

If many substitution rules are defined, it may occur that some of them is applied when you do not expect it (maybe, you do not know even that such substitution exists). To understand what happens, you can use the macro `\mayaDebug` which applies the substitutions to its argument (as if it were the argument of `\mayaGlyph` macro), but instead of drawing the glyph, it just prints the glyph code obtained after the substitutions. This macro does not checks if the obtained glyph code is valid. Neither it checks if primitive glyph names are defined. For example, `\mayaDebug{369<204>xyz}` yields `367xyz`.

6. Colors

By default any glyph is black but its color can be changed by adding a color indicator to the the end of its code. It can be added to the glyph name either

immediately or after the left quote ‘. A color indicator is a chain of letters or digits *ind* declared in a command `\mayaRGB{`*r g b*`}{`*ind*`}`. The meaning of the color components *r*, *g*, *b* (numbers between 0 and 1) is standard, e. g., (1 0 0) is red, (0 1 0) is green, (1 1 0) is yellow etc. If *ind* is a single character, the braces around it can be removed. The indicator ‘`r`’ is predefined for the red.

If a color indicator is attached to a group of glyphs enclosed in parentheses, then the whole group is colored.

Example.

```
\mayaRGB{0 0 0.8}c \mayaRGB{0.9 0.6 0}{orange}
\maya{451.452orange  026.(314/314)c  (570/014‘r.267.024)c}
```

yields  . In the last glyph, the color indicator ‘`r`’ (red) has a higher priority than ‘`c`’ (blue) because ‘`r`’ is inside the parentheses. The glyph `570/014` is colored in red entirely due to the ligature `570/014` → `571`.

The command `\mayaRGB` is retroactive, so, it is recommended to put it near the beginning of your tex file.

When MayaPS interprets a glyph code, first it applies the substitutions and then it tries to extract the color indicator. For example, `\maya{T1c 070:340r}` is interpreted as `\maya{026c 353r}` and yields  .

A more delicate example: the font `codex` has a Thompson code substitutions `T756c` → `454`. Suppose, we define a color indicator `c` as above by the command `\mayaRGB{0 0 0.8}c`. Then `c` at the end of the Thompson code `T756c` is not interpreted as the color indicator. So, the command `\maya{T756c T756cc}` yields  .

When you use the glyph `T756c` in the font `thompson`, then `T756c` is the glyph name, not a substitution. If `c` is defined as a color indicator, then the commands `\maya{T756c}` would provoke an attempt to load a glyph `T756` which does not exist. To avoid this problem one can write `\maya{T756c‘}`.

Remark. Standard LaTeX packages for text coloring do not change the colors of Maya glyphs.

The macro `\mayaIgnoreRGB` (retroactive) cancels the coloring throughout the document. If it is used, then `\mayaRGB{0 0 1}c\maya{026c}` yields  . Attention: if the color indicator ‘`c`’ is not defined at all, then `\maya{026c}` leads to an error ‘glyph not found’.

7. Glyph showing and paragraph formatting

7.1. Macro `\mayaGlyph{`*GlyphCode*`}`

Writes the composed glyph described by *GlyphCode* using the current Maya font of the current font size. Spaces are not allowed between the braces and the glyph code! This command just creates an hbox (see [4; Ch. 11]) with the glyph. For example, if you type `something \mayaGlyph{422.001}\dots`, you obtain:

something  ...

There is a standard problem in TEX with hboxes (it does not appear for the macro '`\maya`'; see §7.4). If you start a paragraph with several hboxes (for example, with several `\mayaGlyph{...}` commands), then TEX puts the boxes one beneath another. To force TEX to leave the vertical mode, you can start the paragraph, for example, with `\hskip0pt` (see [4; Exercise 13.1] for more details).

To be more precise, the argument of `\mayaGlyph` macro is not a glyph code, but a string (chain of characters) which transforms into a glyph code after application of the substitutions and the ligatures (see §5).

If a glyph name is not found in the current font, then `\mayaGlyph` produces something like this 

(for `\mayaGlyph{422.xyz}`). If the parentheses are unbalanced in the glyph code, then it produces 

(for `\mayaGlyph{422.(001}`).

Cartouche. We shall use the term *cartouche* for the box created by `\mayaGlyph` macro. In this subsection, we shall denote its sizes by h (height) and w (width). They are stored in the "hidden" TEX's registers `\maya@xsize` and `\maya@ysize`. The ratio w/h is a fixed for each font. It is called *the aspect* of a font. For the fonts `codex` and `gates` the aspect is $w/h = 23/15 = 1.53333$ (this value is recommended by Bruno Delprat for all Maya fonts).

We do not recommend to change the aspect, however it can be done. For example, after the command `\mayaSetCartoucheAspect{7.5}` the glyphs of the current font become as ugly as this:  .

The recommended way to change w and h is the macro `\mayaSize{`new h`}`, for example, `\mayaSize{5mm}`. The action of this macro is explained below in more detail.

The initial value of h is `12mm`.

How a glyph is placed in the cartouche. A composed glyph is rescaled so that it completely fills the cartouche unless the glyph is obtained by vertical attachment only. In the latter case (for glyphs like `A:B:C`), the glyph is rescaled so that it fills a rectangle $\frac{w+h}{2} \times h$ which is placed in the middle of the cartouche.

A single primitive glyph (even preceded by a modifier) is displayed in its natural proportions (i.e., the ratio height/width is not changed) and it is inscribed into a rectangle $\frac{w+h}{2} \times h$ which is placed in the middle of the cartouche.

Empty argument. When the macro `\mayaGlyph` is called with the empty argument (no spaces between the braces!), it produces an empty cartouche $w \times h$. For example, `abc\mayaGlyph{}\dots` produces

abc ...

(note, that `abc\mayaGlyph{()}\dots` yields abc  ...).

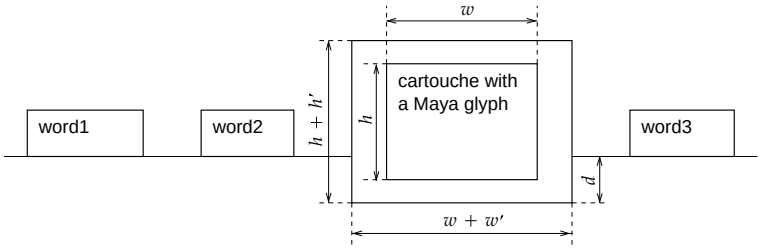
7.2. Macro `\mayaGlyphInLine{`*GlyphCode*`}`

The action of this macro is equivalent to the action of the `\maya` macro with a single glyph code (except that it works a little bit faster, but who cares of it

today). However, it is more convenient to explain `\maya` via `\mayaGlyphInLine`.

The macro `\mayaGlyphInLine` serves to insert maya glyphs into a paragraph written in a European language (as is done everywhere in this text). Its action depends on the current font sizes (of the both current fonts: European and Maya) and it is controlled by the registers `\mayahspace`, `\mayavspace`, and `\mayavcorrection`. Their values are changed automatically by the command `\mayaSize` (see §7.3), but they can be changed manually in the usual way. For example, the command `\mayavspace=2pt` sets a new value of `\mayavspace`, and `\advance\mayavspace by 1pt` increases the value of `\mayavspace`.

The macro `\mayaGlyphInLine` creates a box $(w + w') \times (h + h')$ where w' and h' are the values of `\mayahspace` and `\mayavspace` and $w \times h$ is the current cartouche size. Then it inserts the cartouche made by `\mayaGlyph{`*GlyphCode*`}` in the middle of the box, lowers the box by the value of d (see below), and then TEX formats the paragraph as if it were an ordinary word of the width $w + w'$ (see the figure).



The value of d (the depth) is computed by the formula $d = \frac{1}{2}(h - \frac{3}{2}\texttt{ex}) + \frac{1}{2}h' - c$ where h is the cartouche height, h' is the value of `\mayavspace`, c is the value of `\mayavcorrection`, and `ex` is the height of the letter 'x' in the current Latin font.

Thus, the `\mayavcorrection` parameter is used to move glyphs vertically. For example,

```
Some text \maya{422.001} {\mayavcorrection=1mm\maya{422.001}}
more text \maya{422.001} {\mayavcorrection=-1mm\maya{422.001}}.
```

produces

Some text  more text  .

The parameter `\mayavspace` influences the interline space in the usual way.

7.3. Macro `\mayaSize{` *dimen* `}`

Sets the height h of the cartouche to *dimen*. Here *dimen* is a dimension in the format of TEX or LATEX, for example, `4mm`, `1.2cm`, `0.1in`, `12pt`, `15bp`, etc. (`1in` = 1 *inch* = `72.27pt` = `72bp`).

This command changes the width w of the cartouche proportionally, i.e., (new h)/(new w) = (old h)/(old w), and it sets the size of the caption font.

It sets also `\mayahspace` $= \frac{1}{30}h$, `\mayavspace` $= \frac{1}{10}h$, and `\mayahskip` $= 0$.

7.4. Macro \maya{*GlyphCode*$_1$ *GlyphCode*$_2$ ... *GlyphCode*$_n$}

Writes a sequence of Maya glyphs separated by spaces and newline characters. (the glyphs are formatted by the \mayaGlyphInLine macro). This macro is the main macro of all the package. It is possible to use only it and nothing else. It is forbidden to put anything between the glyph codes (no commas, no dots, no TEX commands). Spaces before the first and after the last glyph code are allowed but ignored. The argument of \maya may have several paragraphs.

The action of \maya is controlled by the parameters from §7.2 and also by the skip (glue) register \mayahskip. If the parameter \mayahskip is set to zero, then \maya{g_1 g_2 ... g_n} is equivalent to

\mayaGlyphInLine{g_1} \mayaGlyphInLine{g_2} ... \mayaGlyphInLine{g_n}.

If \mayahskip is nonzero then its value is added (as an additional horizontal skip) between each pair of consecutive glyphs.

The value of \mayahskip is set automatically by the \mayaSize macro (see §7.3), but it can be redefined in a standard way. For example, by the command \mayahskip=0pt\relax or \mayahskip=1mm plus 0.5mm minus 0.5mm

7.5. Glyphs with Captions

7.5.1. The macros

\mayaGlyph, \mayaGlyphInLine, and \maya

have analogs with *captions*. These are

\mayaGlyphC, \mayaGlyphInLineC, and \mayaC.

For example, if you type

```
\mayahskip=10pt
\mayaC{422.001  321<173>  ?422} some text \mayaGlyphC{T1.001}
```

then you obtain:

So, the caption-macros act by the same algorithms that their no-caption prototypes, but the cartouche is enlarged from the bottom by a caption where the argument of the \mayaGlyph macro is shown.

Double captions. The macro \mayaGlyphCC acts like \mayaGlyphC but the caption has two lines: the first line is the argument and the second line is the result of substitutions (ligatures). Roughly speaking, the second line is the result of \mayaDebug macro (see §5.6). Example:

```
\mayaGlyphCC{369<204>.xyz}\hskip30pt\mayaGlyphCC{570/014.267.024}
```

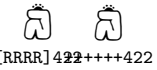
369<204>.xyz 570/014.267.024

yields 367.xyz 571.267.024

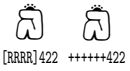
The macro `\mayaGlyphCC` is used for construction of font maps (see §4.2).

7.5.2. Captions of fixed width: `\mayaCFWtrue` and `\mayaCFWfalse`. When the distance between two glyphs with captions is computed, the width of the captions is not taken into account. So, if you type several lines of Maya glyphs using the `\mayaC` command, then the glyphs are automatically aligned.

There are two possible modes of the caption tool in MayaPS: the *fixed width mode* and the *natural width mode*. The fixed width mode is switched on by the command `\mayaCFWtrue` and it is switched off by the command `\mayaCFWfalse`. By default, the fixed width mode is switched on.

The natural width mode may lead to the overlapping of the captions if they are too long: 

. In this case one can increase `\mayahskip` or `\mayahspace`.

If the fixed width mode is on, then wide captions are compressed (only in the horizontal direction) to fit the cartouche width: 

. Example:

```
\mayaCFWtrue  \mayaC{570/014.267.024 111.+176/111} \hskip20pt
\mayaCFWfalse \mayaC{570/014.267.024 111.+176/111}
```

yields 

7.5.3. Parameters. The caption-macros are controlled by the same parameters as their no-caption prototypes (see §§7.1 – 7.3) and also by the dimension register `\mayavspaceC`. Its value is the distance between the cartouche and the caption text. The default value is 3pt.

Example:

```
Some text \mayaC{422.001} { \mayavspaceC=2mm \mayaC{422.001} }
more text \mayaC{422.001} { \mayavspaceC=1pt \mayaC{422.001}}
more text \mayaC{422.001} {\mayavcorrection=2mm\mayaC{422.001}}
```

produces

Some text 

more text 

more text 

7.5.4. Font in captions. The default caption font is `cmtt` scaled proportionally to the cartouche size. For example, `\mayaSize{12mm}` sets `cmtt9` as the caption font. The size of the caption font can be changed by the commands `\mayaCfive`, `\mayaCsix`, . . . , `\mayaCeighteen`. For example,

```
\mayaCfive\mayaC{422}\mayaCseven\mayaC{422}\mayaCnine\mayaC{422}
```

yields: 

These commands set only `\tt` font (typewriter font). To set another font, use

`\let\mayaCaptionFont=\`*cs*

where `\`*cs* is a control sequence associated to any TEX font. For example,

```
\font\f=cmr5 at 15pt
\mayaC{?422} { \let\mayaCaptionFont=\f  \mayaC{422} }
\mayaC{?422} { \let\mayaCaptionFont=\it \mayaC{?422} }
```

yields 

7.5.5. Text in captions. If the caption-macros are used as described above, the text appearing in the caption just coincides with the argument of the macro (as is seen in the examples). Any other text can be inserted by commands `\mayaGlyphC*{`*Text*`}{`*GlyphCode*`}`, `\mayaGlyphInLineC*{`*Text*`}{`*GlyphCode*`}`, `\mayaGlyphCC*{`*Text1*`}{`*Text2*`}{`*GlyphCode*`}`

(no analog for `\mayaC` macro). For example,

```
\mayaGlyphC*{\bf arm}{=314}\mayaGlyphCC*{unreadable}{glyph}{005}
```

yields: 

7.6. Local and global action of commands

Recall (see [4] for more details), that the state of TEX, and hence, its behavior, depends on many parameters (current font, interline space, definitions, etc.). Some commands change the state of TEX. The changes can be *local* or *global*. Local changes are valid only till the end of the group containing the command. A *group* is, roughly speaking, a part of a `tex` file enclosed in braces `{}` (see [4; Ch. 5] for more details). Global changes of TEX's state are valid till the end of the file (or till a command which explicitly cancels the changes). For example, the font change commands `\it`, `\rm`, ..., and the commands `\def`, `\`*register*`=`*value*, have the local action, but the commands `\gdef`, `\global\`*register*`=`*value* have the global action.

Here we list the commands that cause global/local change of MayaPS state.

Global	Local
`\mayaGlobalDefine`	`\mayaDefine`
`\mayaGlobalUndefine`	`\mayaUndefine`
`\mayaFont`	`\`*cs* (defined by `\mayaFont\`*cs*`=`*file*)
`\mayaAddGlyph`(r.a.)`\mayaImport`(r.a.)	`\mayaSize`, `\mayaSetCartoucheAspect`
`\mayaRGB` (r.a), `\mayaIgnoreRGB` (r.a.)	`\mayaCfive`, ..., `\mayaCeighteen`
`\global\mayahskip=`*glue*	`\mayahskip=`*glue*
`\global\`*register*`=`*dimen*	`\`*register*`=`*dimen*
`\global\let\mayaCaptionFont=\`*cs*	`\let\mayaCaptionFont=\`*cs*
`\global\mayaCFWtrue`	`\mayaCFWtrue`

(here `\`*register* is one of `\mayahspace`, `\mayavspace`, `\mayavcorrection`, `\mayavspaceC`; 'r.a.' means 'retroactive').

7.7. Maya glyphs in tables of contents.

If you use LaTeX's macro `\tableofcontents`, then for inserting Maya glyphs to section titles, use **only** the macros `\maya`, `\mayaGlyph`, `\mayaGlyphInLine` preceded by `\protect`. You may use also `\mayaTOC{`g_1 g_2 ... `}` which is equivalent to `\protect\maya{`g_1 g_2 ... `}`. For example:

```
\codex\mayaSize=20pt \tableofcontents
  . . . . .
\gates\mayaSize=30pt \section{All about \mayaTOC{023:023}}
  . . . . .
\section*{All about \maya{422.422}} % \protect not needed here
\addcontentsline{toc}{section}{All about \protect\maya{422.422}}
  . . . . .
```

Note, that the same glyph may be printed in a section title and in the table of contents in different sizes and even in different fonts (as in example above). In fact, the table of contents is created in two passes (you latex your file twice). On the first pass, a temporary file `.toc` is created and if you use `\protect\maya{` ... `}` in a section name, then `\maya{` ... `}` (without `\protect`) appears in the `toc` file. On the second pass, the `toc` file is included by the `\tableofcontents` command, thus the glyph design in the table of contents is governed by the parameters which are current at that point.

The caption versions of these macros do not work properly in the table of contents. In fact, they almost work. The only problem is that the glyphs are displaced horizontally. The displacement can be corrected manually like this:

```
\mayaSize{20pt} \tableofcontents
 . . . .
\section*{All about \mayaC{422.422} and some other glyphs}
\addcontentsline{toc}{section}{All about \hskip-50pt
      \protect\mayaC{422.422} \hskip50pt and some other glyphs}
```

All this concerns also Maya glyphs used in lists of figures, indexes etc.

8. Error diagnostics

8.1. Errors detected by TEX. These are errors appeared, for example, if you misspell a name of a macro, if you forget a brace, etc. The standard TEX/LATEX mechanism of the error diagnostics works in this case.

The most disturbing error is caused by spaces at the end of the argument of `\mayaGlyph`, `\mayaGlyphInLine`, or their caption versions. In the current version of MayaPS, in this case TEX stops with an error message like this

```
! Argument of \maya@sA has an extra }.
<inserted text>
                \par
   . . . . .

1.15 \mayaGlyphInLine{ }
```

It is not so clear from this message that the problem is the space in the argument. Fortunately, the line with the error is indicated correctly ('`1.15`' in this example) as well as the macro which caused the problem. Moreover, if you ignore this message (by keeping to press 'Enter', or by replying `q` to TEX), then you have a good chance to compile correctly the rest of the document.

8.2. Errors detected by a PostScript interpreter (i.e., error messages appeared while printing or viewing the `ps` file or convertion it to pdf). Normally, such errors do not occur unless you use bad `mpf` files or include bad `eps` files by `\mayaAddGlyph` command (or `eps` files not satisfying the restrictions discussed in §4.2). Another exception is mentioned in §4.3.2, Remark 1. If this is not so, then it is a bug and I will be grateful if you inform me about it.

8.3. Errors in glyph codes. See §7.1.

8.4. MayaPS warnings. When something is wrong, MayaPS always tries to go on nonetheless (doing, maybe, not exactly what you want). In this case usually (not always) it writes a warning to the same output stream where TEX writes his messages. Usually, this is the file `.log` and/or the terminal.

8.5. Capacity restrictions. TEX capacity can be exceeded if you use too many Maya fonts. Usually, TEX allows 60,000 multiletter control sequences (names of macros). MayaPS creates some macros for each substitution in Maya fonts and for each used primitive glyph. For example, `codex` creates up to 3300 macros. So, if you use 17 Maya fonts like `codex`, you may get TEX's message

```
! TeX capacity exceeded, sorry [hash size=60000].
If you really absolutely need more capacity,
you can ask a wizard to enlarge me.
```

TEX can also refuse a single macro `\maya` with too many arguments.

PostScript capacity. If you use too many glyphs, you may encounter problems with PostScript virtual memory (VM). For more details see [6], `config.ps` (the line `m`), and comments in `mayaps.tex` about `\mayaNoVMtrick`.

9. How MayaPS works (interaction TEX/Dvips)

In this and in the next sections (§9 and §10) we assume that the reader knows something about PostScript (see [10] for a short introduction and [1] for a complete description of the PostScript language). Understanding of this section is not necessary for usage of MayaPS with existing fonts but it could be helpful for creating new fonts.

9.1. Dvips features used in MayaPS

Dvips creates a `ps` file which has *prolog*, *setup*, and *body*. The body consists of page descriptions which are all independent on each other. In particular, all definitions which are made on one page cannot be used on another page.

The TEX command `\special{header=`*file*`}` makes Dvips to include a file into the prolog of the resulting `ps` file.

The TEX command `\special{!` *PostScript commands*`}` makes Dvips to include the PostScript commands into the prolog of the resulting `ps` file. When the commands are executed, the dictionary `SDict` is on the top of the dictionary stack. Thus, all key-value pairs defined here, are stored in `SDict`.

The TEX command `\special{"` *PostScript commands*`}` makes Dvips to include the PostScript commands into the body of the resulting `ps` file at the place corresponding to the place in the `tex` file where `\special{"...}` occurs. When the PostScript commands are being executed, the dictionary `SDict` is on the top of the dictionary stack and its state is not changed since the last `\special{!...}`. Thus, all key-value pairs defined there are available.

9.2. Rough structure of MPF files A file MPF (MayaPS Font) contains PostScript programs which are (partially) included by Dvips into the resulting `ps` file. An `mpf` file has the following structure:

Font Header
`%@`
Glyph Description Section

The Glyph Description Section has the following structure:

Glyph Description Header
`%@` g_1
Description of g_1
`%@` g_2
Description of g_2
...
`%@`
`end end end end`

where $g_1, g_2, \ldots$ are glyph names (see §3.1). Thus, the description of a glyph g_i is the text between the line '`%@` g_i' and the next line starting with '`%@`'. A complete definition of the MPF format is given in §10.

9.3. How MayaPS works in the standard mode

If everything runs well, then MayaPS works in the following way.

When TEX scans the file, it creates an auxiliary file `mayaps.tmp` and writes the following data into it:

1). At the first moment, it writes a copy of `mayaps.pro` skipping extra spaces and the lines starting with the percent sign (comment lines).

2). Each time when a macro `\mayaFont` (see §4.1) is expanded, the font header (see §9.2) of the corresponding font file is appended to `mayaps.tmp` skipping extra spaces and comment lines. It is placed between the lines

```
userdict begin MayaDict begin tmpini end end
userdict begin MayaDict begin tmpend end end
```

3). At the end of each page MayaPS writes to `mayaps.tmp` the following data. Suppose that glyphs $g_{i_1}, g_{i_2}, \ldots$ (for example, $g_5, g_{13}, \ldots$) of a font F, glyphs $g'_{j_1}, g'_{j_2}, \ldots$ of a font F', etc., appear on this page the first time since the beginning of the document. Then MayaPS writes to `mayaps.tmp`:

`userdict begin MayaDict begin` n d_F `AddGlyphs`

Description of g_{i_1} *from the font* F

Description of g_{i_2} *from the font* F

...

`end end end end`

`userdict begin MayaDict begin` n' $d_{F'}$ `AddGlyphs`

Description of g'_{j_1} *from the font* F'

Description of g'_{j_2} *from the font* F'

...

`end end end end`

and so on...

where $n = 5\times$(the number of newly appeared glyphs from F) and d_F is the *font descriptor* of F, i.e., an integer number assigned to F when the command `\mayaFont` is executed (n' and $d_{F'}$ mean the same for the font F'). When the glyph descriptions are copied into `mayaps.tmp`, extra spaces and comment lines are removed.

4). Each time when the macro `\mayaAddGlyph` is expanded, MayaPS writes to `mayaps.tmp`:

`userdict begin MayaDict begin` 5 d_F `AddGlyphs`

Glyph Description as in §11.3

`end end end end`

and it is supposed from that moment, that the glyph is used.

The file `mayaps.tmp` is included into the prolog of the resulting `ps` file by the command `\special{header=mayaps.tmp}`. When the `ps` file is interpreted

(printed or viewed on the screen), all the glyph definitions from `mayaps.tmp` are done before interpreting the first page of the document. This is an explanation of the retroactivity of the macro `\mayaAddGlyph` (see §4.2).

All the glyph drawing macros of MayaPS call finally the macro `\mayaGlyph`. This command first applies the substitutions (ligatures), then it checks if the parentheses are balanced, and then it issues the command

`\special{"M(`*GlyphCode*`)` w h d `E}`

where $w \times h$ is the cartouche size, d is the descriptor of the current font (see above in this subsection), and *GlyphCode* is just the glyph code in the same syntax as described in §3.1. Thus, all the work of interpreting the glyph code and drawing the glyph according to §3 is delegated to a PostScript interpreter (i.e., it is postponed till the moment of printing, viewing, or conversion of the `ps` file). This work is done by the PostScript procedure `E` which is defined in the file `mayaps.pro` and stored in the dictionary `MayaDict`. The latter is opened by the procedure `M` which is put to the dictionary `SDict` by the command `\special{!/M{...}def}`.

9.4. Work in absence of some files

If `mayaps.pro` or an MPF file is not found by TEX, then TEX issues the command `\special{header=...}` assuming that the file will be found by Dvips. If a font is not found by TEX but it is found by Dvips, then the aspect and the ligatures defined in the font file are ignored.

The format MPF presumes that all fonts work properly in this mode.

10. MPF format description

Recall that we assume in this section that the reader knows the PostScript language at least as much as is written in [10]. In the next two sections we explain how to create a Maya font out of `eps` files without any knowledge of PostScript.

10.1. More about glyph names and types. There is no formal difference between the names of affixes and those of central elements. In contrary, if it is already known that a given glyph is an affix, then the property to be a numeral is prescribed by the first character of its name.

A list of Numeral's Initial Characters is associated to each Maya font. If the initial character of an affix belongs to this list, then the affix is a numeral. Otherwise it is non-numeral. For the font '`codex`' this list consists of two elements: '`8`' and '`9`'.

10.2. General Structure. Comments. A file `mpf` (MayaPS Font) contains PostScript programs which are included by Dvips into the `ps` file as described in §9. It consists of two sections:

- **Font Header** (see §10.3).
- **Glyph Description Section.** (see §10.4).

Empty lines are in `mpf` files are ignored. An `mpf` file may contain comments. They start with the percent sign `%`, i. e., everything between '`%`' and the end of the line is ignored. If '`%`' is the first symbol of a line, then the line is not included by MayaPS into the resulting `ps` file. Otherwise a comment is transmitted to the `ps` file.

Comments may not start with the combinations `%@`, `%M@`, `%L@` `%V@` and `%W@` because they are reserved for the information exchange between `tex` and `ps`. Comment lines which start with `%`c`@` where c is any character, are reserved for future versions of MayaPS.

Lines starting with `%V@` and `%M@` are used only in font maps (see §4.2) as version information and section names. For example, the text 'v 0.35 (Oct 27, 2012' in the title and the section name 'Irregular Affixes' in §4.2 are defined in `codex.mpf` by the lines

```
%V@ version 0.35 (Oct 27, 2012)
%M@ Irregular Affixes
```

Remark. In §10.3 and §10.4, when we explain the action of PS commands from an `mpf` file, we assume for simplicity that the `mpf` file is included by the command `\special{header=`$fontname$`.mpf}` (see §9.4).

10.3. Font Header.

Contains definitions of variables and procedures (key-value pairs) used in the glyph descriptions. It starts with the line

`userdict begin MayaDict begin (`s`)` n `NewFont`

followed by the *Header Body*, and terminates by the two lines

```
end end end
%@
```

Here `(`s`)` is the string of Numeral's Initial Characters (see §10.1), for example, in '`codex.mpf`' its value is `(89)`, and n is the number of key-value pairs defined in the header body. The command `NewFont`, in particular, creates two new dictionaries associated to the font (let us denote them here by D_1 and D_2) of capacity 1 and n respectively. It makes the dictionary D_2 current (the dictionary stack becomes: ... `userdict MayaDict` D_2). The dictionary D_1 (resp. D_2) is stored as the i-th entry of the array `GDicts` (resp. `GGDicts`) where i is the number associated to the font at the moment of expansion of the macro `\mayaFont`. The arrays `GDicts` and `GGDicts` are stored in the dictionary `MayaDict`.

The *Header Body* may contain definitions which are put into the current dictionary D_2. All standard PostScript commands are available here. The dictionary D_2 will be used in glyph descriptions (see the next subsection). For example, the font header may look like this (the empty header body):

```
userdict begin MayaDict begin (89) 0 NewFont
end end end
%@
```

(then only standard PS commands will be available in the glyph descriptions).

A font header may contain an aspect declaration (see §7.1). It is a line

`%W@` a

where a is any number. The default value of aspect is $23/15 = 1.53333$.

A font header may contain also any number of substitution (ligature) declarations. These are lines of the form

`%L@` s_1 s_2

(at least one space between `%L@` and s_1, and at least one space between s_1 and s_2). Such a line acts as the command `\mayaDefine{`s_1`}{`s_2`}` described in §5.3. Substitution declaration lines may be placed everywhere before the end-of-header line '`%@`'.

10.4. Glyph Description Section.

This section starts with the line

`userdict begin MayaDict begin` k `AG`

followed by any number of *Glyph Descriptions*, and terminates by the two lines

```
%@
end end end end
```

The parameter k should be greater than the number of key-value pairs defined in all the glyph descriptions. For example, it is always possible to set $k = 1+5\times$(the number of glyphs). The command `AG` increases the capacity of D_1 by k and it leaves D_2 D_1 on the top of the dictionary stack (after this command, the dictionary stack becomes: ...`userdict MayaDict` D_2 D_1).

A *Glyph Description* consists of a line

`%@` *name*

followed by a *Glyph Description Body*. The latter must contain definitions of the following key-value pairs (and nothing else):

- `w`*name* (number) width of glyph;
- `h`*name* (number) height of glyph;
- `a`*name* (boolean) true if affix;
- `A`*name* (array; optional) `[(`L`)(`U`)(`R`)(`D`)(`S`)]` where $L, \ldots, S$ are strings of modifiers (see §3). The default value: `[()(|r)(|)(R)()]`.
- `m`*name* the procedure of glyph drawing. It should have the form `{GIni` *PostScript commands* `GEnd}` (see Remark 2 below). The procedures `GIni` and `GEnd` contain a gsave-grestore pair, but they do not contain any save-restore pair. So, the procedure should not leave anything on stacks.

The dictionary D_2 is open at the moment of the definition of these objects (as we told already, it is open by the command `AG`). It will be also open when the procedure `m`*name* will be executed (the command `GIni` opens it). So, the names defined in D_2 may be used everywhere. The procedure `m`*name* should draw the glyph in the bounding box `[0 0` w h`]` where w and h are the values of `w`*name* and `h`*name*.

Remark 1. Only the ratio h/w is important, but not the values of w and h themselves. For example, if you multiply w and h by 2 and insert the command `2 2 scale` just after `GIni`, then the result will be the same.

Remark 2. If you do not use the definitions from D_2, then it is not necessary to include the GIni-GEnd pair, but in this case you are responsible yourself for not to change the graphic state.

Remark 3. As we explained already, `AG` leaves `...userdict MayaDict` D_2 D_1 on the dictionary stack. Thus, the names defined in D_2 are available at the moment of definition of `w`*name*, `h`*name*, `a`*name*, etc. This means, for example, that a definition '`a`*name* `false def`' can be abbreviated up to '`a`*name* `F D`' if the names `F` and `D` were defined in the dictionary D_2 by the commands '`/D{def}bind def/F false D`' in the font header (this possibility is used in fonts of §11.5). Such names (like `F` and `D` in our example) should not start with `w`, `h`, `a`, `A`, and `m` if they consist of more than one character. This is a precaution to avoid conflicts with the names from D_1. Even if you know that a glyph name is not used in the font, a user can introduce it by `\mayaAddGlyph` macro.

10.5. Examples.

Example 1. Suppose that an `mpf` file is:

```
% Empty MayaPS Font
userdict begin MayaDict begin (89) 1 NewFont
%@
end end end
% end of the font header
userdict begin MayaDict begin 0 AG
%@
end end end end
```

Then it is a valid font which has no glyphs. Such a font is not as useless as one could think. One can define glyphs using `\mayaAddGlyph` macro.

Example 2. Suppose that the file `fractal.mpf` contains:

```
userdict begin MayaDict begin () 15 NewFont
/D{def}def/B{bind def}D/d{dup}D/gs{gsave}D/gr{grestore}D
/t{d translate}D/s{scale}D/r{rotate}D/w{setlinewidth}D
/C{curveto}D/y{58}D/m{newpath moveto}B/sb{add r neg 100 add}B
/f{2 w 30 2 m 2 2 2 2 2 30 C 2 y 2 y 30 y C
   y y y y y 30 C y 2 y 2 30 2 C stroke}B
/F{2 w .5 -.5 m 30 30 lineto -.5 .5 lineto stroke
```

```
  d 0 gt{7 8 23{gs d t d 280 sb 210 div d s d 1 sub F gr}for
         9 8 25{gs d t d neg 80 sb 230 div d s d 1 sub F gr}for
         gs 28 t -5 r .4 d s d 1 sub F gr}if pop}B
end end end
%@
userdict begin MayaDict begin 20 AG
%@ 0
/w0 60 D/h0 60 D/a0 false D/m0{GIni f 9 t 0 F GEnd}B
%@ 1
/w1 60 D/h1 60 D/a1 false D/m1{GIni f 9 t 1 F GEnd}B
%@ 2
/w2 60 D/h2 60 D/a2 false D/m2{GIni f 9 t 2 F GEnd}B
%@ 3
/w3 60 D/h3 60 D/a3 false D/m3{GIni f 9 t 3 F GEnd}B
%@ 4
/w4 60 D/h4 60 D/a4 false D/m4{GIni f 9 t 4 F GEnd}B
%@
end end end end
```

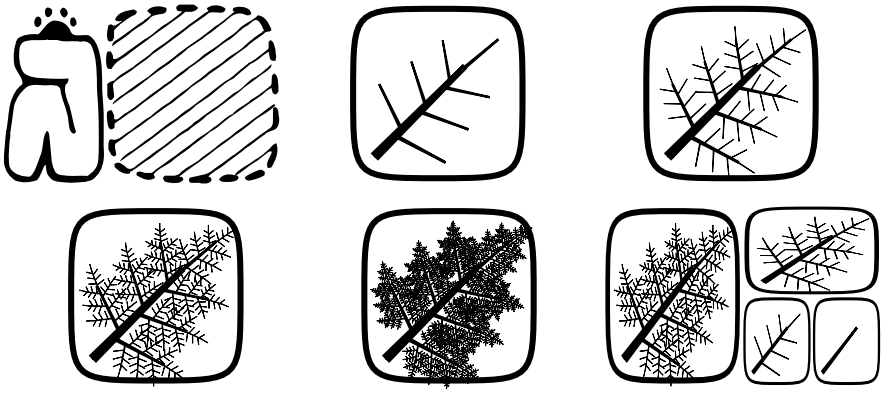
and a `tex` file contains

```
\input maya
\mayaFont\fractal=fractal
\mayaSize{24mm}
\mayahskip=-3pt\relax\mayahspace=-3mm
Fonts {\tt codex} and {\tt fractal}:

\noindent\maya{422.001} \fractal \maya{1 2 3 4 3.2:(1.0)}
\end
```

Then plain TEX (not LATEX) and Dvips give the output:

Fonts `codex` and `fractal`:



Exercise. What is the Hausdorff dimension of the glyph `\maya{`∞`}`? `:-)`

11. How to make a Maya font out of EPS files

No knowledge of PostScript Language is required for understanding this section. All necessary information about the EPS format is given in §11.2.

We assume that a user has a collection of EPS (Encapsulated PostScript) files with glyph images and we explain how to create an MPF (MayaPS Font) file out of them by copying some parts of EPS files into the MPF file. Since the both EPS and MPF files are ASCII text files (at least those EPS files which are allowed here; see the next subsection), this can be done, for example, by a straightforward 'copy-paste technology' using any text editor (which is rather boring however). Otherwise, it is easy to write simple programs (scripts) which do this more or less automatically.

11.1. Restrictions on EPS files.

We assume that EPS files do not contain the backslash character '\' (compare with §4.2). We assume also that EPS files do not contain 'unprintable' characters except the tabulation character and all kinds of end-of-line characters (usually they don't).

11.2. On the structure of EPS files.

An EPS (Encapsulated PostScript) file is a PS program satisfying some additional conventions which ensure that any such file can be included into any PostScript document. An EPS file starts with the 2 characters '`%!`' followed by an information on the version of EPS format (in the first line). Next several lines start with the double percent sign '`%%`' These are *DSC header comments* (DSC means Document Structuring Conventions). One of them must contain the *bounding box* – the rectangle limiting the picture. This is a line of the form

`%%BoundingBox:` $llx\ lly\ urx\ ury$

where (llx, lly) are the coordinates of the lower-left corner of the box and (urx, ury) are the coordinates of the upper-right corner. The coordinates are given in big points (`bp`, see §7.3) but the unit of measure is not important for us, because each glyph will be scaled anyway to the size defined in a `tex` file.

In old versions of EPS format, the bounding box line was allowed to be placed near the end of file.

These two lines (the `%!`-line and the bounding box) are the only required DSC header comments in an EPS file. However, the most of programs producing EPS files include more DSC header comments. The last one is

```
%%EndComments
```

Usually EPS files are organized as follows. They begin with a *prolog* (the common part of all EPS files produced by the given program) followed by a *script* (the part which varies from one EPS file to another). Usually, the prolog is written by a programmer and the script is generated automatically by the program. The prolog ends with the line

```
%%EndProlog
```

11.3. The simplest (not the best) way to make an MPF. An MPF file may be composed as follows (see in §10.2 about comments and §10.3 about ligatures).

```
% Title (optional): the font name, the creator, the version, etc.
userdict begin MayaDict begin (s) 0 NewFont
end end end
% Table of Ligatures (Substitution):
%L@ s11 s12
%L@ s21 s22
 ...
%@
userdict begin MayaDict begin k AG

Glyph Description 1

Glyph Description 2
  ...
%@
end end end end
```

where s is the string of Numeral's Initial Characters (see §10.1; for example, `(`s`)` is `(89)` in `codex.mpf`) and k may be set to $5 \times$ (the number of glyphs) $+ 1$ (see §10.4 for more details). Each Glyph Description has the following structure:

```
%@ name
/wname w def /hname h def /aname bool def
/Aname[(L)(U)(R)(D)(S)]def    (optional; only for affixes)
/mname{save
−llx −lly translate              (not needed if llx = lly = 0)
      A copy of the EPS file of the glyph
restore}bind def
```

where: *name* is a glyph name, $w = urx - llx$, $h = ury - lly$, *bool* is `true` if the glyph is an affix and `false` if it is a central element, $L, \ldots, S$ are the orientations as in §3.3 (the default value is `[()(|r)(|)(R)()]`), and $[llx, lly, urx, ury]$ is the bounding box of the EPS file (see §11.2).

Note, that almost all programs creating EPS provide $llx = lly = 0$. In this case the line '$-llx$ $-lly$ `translate`' is not needed and $w = urx$, $h = ury$.

Remarks. 1. Usually EPS files have a rather long header (prolog) which is common for all of them. If a font is created as described here, then MayaPS includes into the resulting `ps` a copy of the prolog for each glyph used in the text. If the text contains several hundreds of primitive glyphs, then the resulting `ps` file could get several useless Mb.

2. If you use an `mpf` file created as described here, then the result will be absolutely the same as if you define all the primitive glyphs used in the text by

`\mayaAddGlyph` macro. An advantage of the latter approach is that you don't care of boundary boxes (`\mayaAddGlyph` does it itself). A disadvantage is that you have to select yourself the glyphs which are used in the text. Otherwise (for example, if you create a 'pseudofont' which can be loaded by the '`\input`' command and which contains many `\mayaAddGlyph`'s), the resulting `ps` file may be huge (see the previous remark).

11.4. Case of EPS files created in a uniform way. In this subsection we assume that all EPS files are created by the same program, For example, by the same graphical editor, or they are scanned and then vectorized by the same vectorizer. More precisely, we assume that all EPS files have the same prolog, i.e., all of them look like this:

The common prolog of all EPS files
`%%EndProlog`

The script (depends on the EPS files)

Then the MPF file can be composed as follows:

`%` *Title (optional): the font name, the creator, the version, etc.*
`userdict begin MayaDict begin (`s`) 1 NewFont`
`/prolog{save userdict begin`
The common prolog without DSC Header Comments
`}bind def end end end`
`% Table of Ligatures (Substitution):`
`%L@` s_{11} s_{12}
`%L@` s_{21} s_{22}
...
`%@`
`userdict begin MayaDict begin` k `AG`

Glyph Description 1

Glyph Description 2

...
`%@`
`end end end end`

where `(`s`)` and k are the same as in §11.3. Each Glyph Description has the following structure:

`%@` *name*
`/w`*name* w `def /h`*name* h `def /a`*name bool* `def`
`/A`*name*`[(`L`)(`U`)(`R`)(`D`)(`S`)]def` *(optional; only for affixes)*
`/m`*name*`{GIni prolog`

$-llx$ $-lly$ `translate` *(not needed if $llx = lly = 0$)*

The script of the EPS file of the glyph

`end restore GEnd}bind def`

where all the parameters are the same as in §11.3.

11.5. MPF file created using `cotrace -n`.

If the `eps` files are created by the vectorizer `cotrace` with the option `-n` (this option ensures that there are no backslash characters '\' in them), a file MPF can be composed as follows.

`%` *Title (optional): the font name, the creator, the version, etc.*
`userdict begin MayaDict begin (`s`) 10 NewFont`

The prolog produced by `cotrace` *with* `-n` *option*

`end end end`
`% Table of Ligatures (Substitution):`
`%L@` s_{11} s_{12}
`%L@` s_{21} s_{22}
...
`%@`
`userdict begin MayaDict begin` k `AG`

Glyph Description 1

Glyph Description 2

...
`%@`
`end end end end`

and the glyph descriptions are:

`%@` *name*
`/w`*name* w `D/h`*name* h `D/a`*name* *bool* `D`
`/A`*name*`[(`L`)(`U`)(`R`)(`D`)(`S`)]D` *(optional; only for affixes)*
`/m`*name*`{GIni`

The script of the EPS file of the glyph

`end restore GEnd}B`

The file `codex.mpf` used in Version 0.23 of MayaPS have this structure.

11.6. Case of EPS files of different origins.

Here we assume that you have a collection of EPS files with one prolog, another collection with another prolog, etc. Let us denote the prologs by $P_1, P_2, \ldots, P_n$. In this case the MPF file described in §11.4 should be changed as follows.

1). The part

`userdict begin MayaDict begin (`s`) 1 NewFont`

`/prolog{save userdict begin`

The common prolog without DSC Header Comments

`}bind def end end end`

should be replaced with

`userdict begin MayaDict begin (`s`)` n `NewFont`
`/prolog1{save userdict begin`

Prolog P_1 *without DSC Header Comments*

`}bind def`
`/prolog2{save userdict begin`

Prolog P_2 *without DSC Header Comments*

`}bind def`

...

`/prolog`n`{save userdict begin`

Prolog P_n *without DSC Header Comments*

`}bind def end end end`

2). '`prolog`' should be replaced by '`prolog`i' in each definition of `m`*name*. Here i is the number of the prolog used in the EPS file of the glyph.

Of course, the names `prolog1`, `prolog2`, ... may be replaced by any other names (except '`GEnd`'), for example, it is reasonable to choose names explaining the origin of the EPS files.

12. Vectorizer CoTrace

12.1. `cotrace` (Compact Trace) is a vectorizer that I wrote for making EPS files of glyphs for the font '`codex`' out of scanned hand-made pictures of Maya glyphs (prepared by Bruno Delprat). However, it can be used in any other situation when a conversion (monochrome bitmap)→EPS is needed.

The main (and, maybe, the only) advantage of `cotrace` with respect to other vectorizers which I tried, is that it produces very short `eps` files.

Instructions for installation and usage can be found in the file `README`.

The program `cotrace` with `-c` option (version 1.1 and higher) produces a glyph description in Type 1 font format. This option is used in the script `makefont` described in the next subsection.

Digression on Adobe Type 1 fonts. PostScript is a powerful programming language. In particular, it allows subroutines. So, it is possible to write a subroutine of drawing a picture and then to call it many times. This is why a PS document containing 1000 identical copies of a picture is not much longer than a document containing it only once. This mechanism is used in MayaPS fonts.

PDF inherited some features of PostScript language but not subroutines. When a PS file with 1000 copies of a picture is converted into PDF, the program of drawing the picture is reproduced 1000 times. The only exceptions are pictures of characters of Type 1 fonts (see [2]).

12.2. Creating a MayaPS font using `cotrace` and `makefont`.

Shell script `makefont` was used for creating the font '`thompson2.mpf`'. It can be easily adapted for creating other fonts. It runs under UNIX/Linux/MacOS.

To create a new MayaPS font, do the following:

1). Prepare rasterized images (bitmaps) of glyphs, for example, by scanning hand-written pictures. The resolution (the size of the image measured in pixels) must be chosen so that all black pixels are contained in a square 3527×3527 pixels (the restriction imposed by `cotrace`). The linear sizes of bitmaps used in the font '`codex`' are about 300–1000 pixels. If you use a higher resolution, you may get a higher quality, but you pay for it by increasing of the length of the EPS file. The file length is proportional to the linear size of a glyph measured in pixels.

2). Make sure that central elements appear in the desired orientation. If the default orientations of an affix are like this

$L =$  504.001 $U =$  504:001 $R =$  001.504 $D =$  001:504 $S =$  504

then it is reasonable to choose the orientation in the bitmap file so that it coincides with S- (the same as L-) orientation of the primitive glyph (like this  for the glyph `504`). In this case you needn't write explicitly the `[(`L`)(`U`)(`R`)(`D`)(`S`)]` vector.

Clean the images using some graphical editor for rasterized graphics and convert (export) the files to the monochrome (1 bit per pixel) BMP format. Under Linux, the conversion to the monochrome BMP can be done, for example, by the program `convert -monochrome` from the package `ImageMagick`. The BMP files should be named *name*`.bmp` where *name* is the glyph name. Put all these file to a zip-archive `bmp.zip`.

3). Create a new directory (say, `make`*fn* where *fn* is the name of the Maya font you are making). Compile the C programs and make the shell scripts executable. It can be done by the following commands:

```
gcc cotrace.c -o cotrace
gcc addg.c -o addg
chmod +x addglyph
chmod +x makefont
```

Copy the files `cotrace`, `addg`, `addglyph`, `makefont` (all without extensions), and `bmp.zip` to the directory `make`*fn*. Modify the file `makefont` which is more or less self-explaining, and run

```
./makefont
```

13. How to make a Maya font out of an Adobe Type 1 font

In this section we suppose that you have an Adobe Type 1 font program (a `pfa` file; see [2]) prepared, e. g., with FontForge, and we explain how to create a MayaPS font out of it.

Remark 1. (Important!) The type 1 font should not use subroutines.

Remark 2. Quotation from [2], page 23: "*. . . assume the 1000 to 1 character space to user space scaling that is typical of the Type 1 font format*". We do the same. Otherwise minor changes are required everywhere including the header.

For simplicity, we use the example considered in [2; §6.6, §7.3]. So, suppose that your Type 1 font has a character named C looking like ⊏ and encoded as in §6.6 and §7.3 of [2]. Here is an example of a MayaPS font (`T1example.mpf`) with this character:

```
%  MayaPS font   T1example.mpf
%  An illustration of pfa->mpf conversion
%
%%% The header for MayaPS fonts based on Adobe Type 1 fonts %%%
```

Omitted PostScript commands can be copied from `codex.mpf`

```
%%% End of the standard part of the header %%%
%  Substitutions (ligatures)
%L@ 555 C
%@
%@ C
/hC 1000 D % should be set to 1000 (see Remark 2 above)
/wC  800 D % 800 is from '50 800 hsbw...', see T1Format.pdf; p.58
/aC F D    % central element (use T instead of F for affixes)
P/mC{GIni()A( )Z}U/gC   % Do not forget P and the space in 'A( )'
<10bf31704fab5b1f03f9b68b1f39a66521b1841f14
81697f8e12b7f7ddd6e3d7248d965b1cd45e2114> %see T1Format.pdf; p.65
V
%@
end end end end
```

Then the commands

```
\mayaFont\testfont=T1example
\testfont\maya{C  |C:?C  +555:(C.*C).C}
```

produce ⊏ ⊐ ⊐⊏

Now we explain how to extract the needed information from a `pfa` file.

1). Extract the eexec-encrypted part (thousands of hexadecimal digits between the command `eexec` and several lines of zeros). Decrypt it using the algorithm

in §7.1, page 63 of [2] with $R = 55665$ (decimal) and $n = 4$ (i.e. discard the first 4 bytes after the decryption). You obtain an ASCII text (PostScript commands) which contains some binary segments enclosed between '`RD`' and '`ND`'.

2). Find the command `/lenIV` n `def`. This value of n will be used later for the charstring decryption in Step 5. If this command is absent, set $n = 4$.

3). Find the command `/`*name* m `RD` *–m–binary–bytes–* `ND` (in the example with ⊏ this is `/C 41 RD` *–41–binary–bytes–* `ND`). There is one space (the code 32) after '`RD`'. It should not be included to the m binary bytes.

4). Rewrite the m binary bytes in the hexadecimal form ($2m$ hexadecimal digits), and put them to the `mpf` file between '`g`*name*`<`' and '`>V`'.

5). The glyph height is always 1000 (see Remark 2). If you don't know the width, decrypt the m binary bytes using [2; §7.1] with $R = 4330$ and n from Step 2; decode the result according to [2; §6] till the command `hsbw` (usually it is the first command) and see the preceding number.

References

[1]. Adobe System Inc. *PostScript Language Reference Manual.* Files `plrm.pdf`, `plrm2.pdf` available on `http://www.adobe.com`

[2]. Adobe System Inc. *Adobe Type 1 Font Format.* File `T1Format.pdf` available on `http://www.adobe.com`

[3]. E. V. Evreinov, Yu. G. Kosarev, B. A. Ustinov, *Using computers for Study of the Ancient Maya Written Language.* Siberian Branch of the Acad. Sci. USSR, Novosibirsk, 1961 (in Russian).

[4]. D. E. Knuth. *The TEXbook.* Addison-Wesley, Boston, 1984.

[5]. L. Lamport. *LATEX: A Document Preparation System* (2nd ed.), Addison-Wesley, Boston, 1994.

[6]. T. Rokicki. *Dvips: A DVI-to-PostScript Translator.* The file `dvips.pdf` included in the most of TEX distributions; available on `http://www.ctan.org`

[7]. A. Syropoulos. *Typesetting Native American Languages.* Journal of Electronic Publishing, **8**(2002), issue 1. `http://www.press.umich.edu/jep`

[8]. *A directory structure for TeX files.* `http://www.ctan.org`

[9]. J. E. S. Thompson, *A Catalog of Maya Hieroglyphs*, Univ. Oklahoma Press, 1962

[10]. I. Utting. *A PostScript Tutorial and Reference.* Available on `http://www.cs.kent.ac.uk/publ/1991/109`

E-mail: `orevkov@math.ups-tlse.fr`

`http://picard.ups-tlse.fr/~orevkov/mayaps.html`