

# MayaPS: Typing Maya with TeX/LaTeX.

## Reference Manual. Version 0.23 (May 18, 2007)

---

Stepan Orevkov

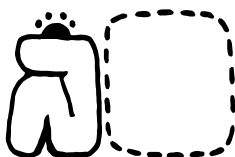


I have written the system MayaPS in collaboration with Bruno Delprat who is its first user. He formulated the principles of ancient Maya typesetting and he composed the list of Maya glyphs. The idea to use PostScript language for drawing composed glyphs belongs to Ilya Zakharevich.

### Table of Content

1. What is MayaPS for?	2
2. Installation and running.	5
2. How to work without any installation.	5
2. Don't use PdfTeX with MayaPS. Use TeX/Dvips.	5
3. Glyph codes and glyph orientations.	7
3.1. Glyph codes.	7
3.2. Glyph types.	7
3.3. Glyph orientations.	7
3.4. Modifiers.	9
4. Usage of different Maya fonts.	10
4.1. Fonts (the macro <code>\mayaFont</code> ).	10
4.2. The macro <code>\mayaAddGlyph</code> .	10
5. Ligatures and substitutions.	12
5.1. Ligatures.	12
5.2. Different encodings (Catalogs).	13
5.3. Substitutions.	13
5.4. Definition/canceling of substitution rules.	14
5.5. Red numerals in the font 'codex'.	14
5.6. The macro <code>\mayaDebug</code> .	15
6. Glyph showing and paragraph formatting.	16
6.1. The macro <code>\mayaGlyph</code> (cartouche).	16
6.2. The macro <code>\mayaGlyphInLine</code> ( <code>\mayahspace</code> , <code>\mayavspace</code> , <code>\mayavcorrection</code> ).	17
6.3. The macro <code>\mayaSize</code>	18
6.4. The macro <code>\maya</code> (also <code>\mayahskip</code> ).	18
6.5. Glyphs with captions.	18
6.5.1. <code>\mayaGlyphC</code> , <code>\mayaGlyphInLineC</code> , <code>mayaC</code> .	18
6.5.2. Restrictions.	19

6.5.3. Parameters ( <code>\mayavspaceC</code> ).	19
6.5.4. Font in captions ( <code>\mayaCfive</code> , <code>\mayaCsix</code> , ..., <code>\mayaSixteen</code> ; <code>\mayaSixteen</code> ; <code>\mayaCaptionFont</code> ).	19
6.5.5. Text in captions ( <code>\mayaGlyphC*</code> , <code>\mayaGlyphInLineC*</code> ).	20
6.6. Local and global action of commands.	20
7. Error diagnostics.	21
8. How MayaPS works (interaction $\TeX$ /Dvips).	22
8.1. Dvips features used in MayaPS.	22
8.2. Rough structure of MPF files.	22
8.3. How MayaPS works. The standard mode.	23
8.4. The mode <code>\maya@TmpFilefalse</code> .	24
9. MPF format description.	24
9.1. More about glyph names and types.	24
9.2. Introduction.	25
9.3. Font header.	25
9.4. Glyph description section.	26
9.5. Examples.	27
10. How to make a Maya font out of EPS files.	29
10.1. Restrictions on EPS files.	29
10.2. On the structure of EPS files.	29
10.3. The simplest (not the best) way to make an MPF.	30
10.4. Case of EPS files created in a uniform way.	31
10.5. MPF files created using <code>cotrace</code> .	32
10.6. Case of EPS files of different origins.	32
11. Vectorizer CoTrace.	33
11.1. <code>cotrace</code> .	33
11.2. Creating a MayaPS font using <code>cotrace</code> and <code>makefont</code> .	33
12. Ideas for future versions of MayaPS.	35
References.	36



## 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 picture, I typed “`\centerline{\mayaSize{2cm}\mayaGlyph{422.001}}`”

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}
```

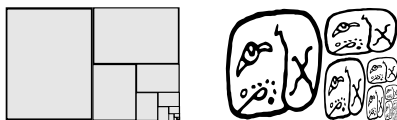
As it should be clear already, each primitive glyph is referred to by its name (called further the *glyph name*). So, the glyph names used above are “422”, “001”, “023”, “153”, and “220”. In general, a glyph name is any sequence of digits 0...9 and letters a...z, A...Z. The encoding system is rather 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








(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 adopted 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 §11.

If  $\boxed{A}$  and  $\boxed{B}$  are two glyphs (primitive or not), then  $A.B$  and  $A:B$  encode the glyphs  $\boxed{A \ B}$  and  $\boxed{\begin{smallmatrix} A \\ B \end{smallmatrix}}$ . 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  $\boxed{\begin{smallmatrix} A & B \\ C \end{smallmatrix}}$  but  $(A.B):C$  stands for  $\boxed{\begin{smallmatrix} A & B \\ C \end{smallmatrix}}$  (in glyph codes, ‘/’ means the same as ‘:’). Any formula of this kind is admitted, even something like this



Here the right picture (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:  $\boxed{\begin{smallmatrix} \text{glyph} \\ \text{glyph} \end{smallmatrix}}$  and  $\boxed{\text{glyph} \ \text{glyph}}$ . 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.

MayaPS provides a tool to add or replace glyphs in existing fonts (this is very easy). In §10 and §11 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  $\text{\TeX}/\text{Dvips}$ ) in a European language, and you include ancient Maya glyphs into it. Then MayaPS adds to ps only:

- MayaPS header (7 Kb);
- definitions of primitive glyphs (300–700 bytes 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 on any modern printer or viewed on the screen using, e.g., `ghostview` or `gv` (Unix/Linux-X11), `GSview` (Windows), etc. The same programs can be used for the conversion `ps`  $\rightarrow$  `pdf` (this can be done also online in <http://www.ps2pdf.com>). The resulting `pdf` file may lose the length optimization (see §12.5).

**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. So, if you use MayaPS, don't be afraid to lose your work when new generations of computers come.

Another attempt to adopt `TEX`/`LATEX` for Native-American (including Ancient Maya) languages was done in [6]. Our approach is very different from that in [6].

## 2. Installation and running

### 2.1. How to work without any installation

Just copy the files `'mayaps.tex'`, `'mayaps.pro'`, and `'codex.mpf'` 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` command, 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 this line just after `\documentclass` command.

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 some graphic interface environment, you should tune it so that it calls `TEX` or `LATEX` and `Dvips`.

### 2.2. Don't use PdfT<sub>E</sub>X with MayaPS. Use T<sub>E</sub>X/Dvips.

**T<sub>E</sub>X, Dvips, and PdfT<sub>E</sub>X.** `TEX` is a program written by Donald Knuth (see [2]). 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`  $\rightarrow$  `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 adopted for modern realities. For example, you can see PDF files directly from Internet browsers, it is interactive, 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`  $\rightarrow$  `dvi`  $\rightarrow$  `ps`  $\rightarrow$  `pdf` is always possible, but Han The Thanh made a shortcut. He has written Pdf $\TeX$  – a program which produces a PDF file directly from `tex` source. It has many advantages, especially concerning a preparation of interactive documents. Many modern  $\TeX$  installations call Pdf $\TeX$  instead of  $\TeX$  by default.

Pdf $\TeX$  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 from primitive ones are implemented on the PostScript programming language.

$\LaTeX$  is an extension of  $\TeX$  written by Leslie Lamport (see [3]) and further developed by the  $\LaTeX$ 3 project team. Any modern installation of  $\TeX$  includes it. MayaPS is compatible with  $\LaTeX$  (this text is prepared with  $\LaTeX$ ). Of course, I could not check the compatibility of MayaPS with all  $\LaTeX$  packages, but I hope that it is compatible with (most of) them.

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

- 1). I enjoy PostScript programming but I don't enjoy PDF programming.
- 2). I don't believe that MayaPS will have more than 100 users for all times (even this is a very optimistic estimate). Half of them will have already Dvips. Suppose that each of the others will spend 1 hour average learning how to switch from Pdf $\TeX$  to  $\TeX$ /Dvips. This is much less than the time I need to write MayaPDF.

- 3). The interaction between  $\TeX$  and Dvips is used not only in MayaPS. There are many other applications which use this mechanism. The Dvips interface is partially implemented in many Dvi viewers such as `xdvi`, `kdvi` (for Unix/Linux-X11), `yap` (for Windows), etc. There are no principal obstacles to incorporate Dvips' interface into Pdf $\TeX$  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  $A, B, \dots$  are glyph names (i.e., names of primitive glyphs), maybe preceded by modifiers ('!', for example) which allow to change the orientation. Let us give formal definitions.

*Glyph Name* (called also *primitive glyph code*) is any sequence of digits 0...9 and letters a...z, A...Z.

*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 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.

**Remark.** The knowledge of the fact that a glyph name is any sequence of letters and digits is helpful only for those users who create a new font (see §§9–11) or introduce a new primitive glyph by `\mayaAddGlyph` command (see §4.2). If you use only existing glyphs in existing fonts, then glyph names are just names (codes) of primitive glyphs in the fonts you are using.

**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.

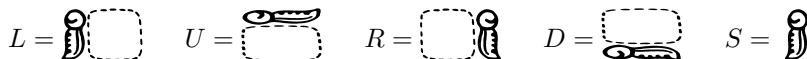
**3.3. Glyph orientations.** Here we describe the default rules for the choice of orientations of primitive glyphs. Here ‘default rule’ means a rule which is applied when a glyph code has no modifiers. These rules are proposed by Bruno Delprat after a careful analysis of ancient Maya manuscripts.

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.

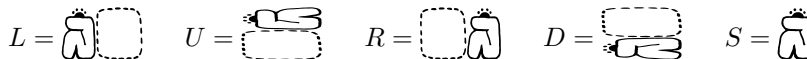
**Default rules for the orientations of primitive glyphs.**

1). Any central element always appears in the same orientation which we call *natural*.

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* is 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):

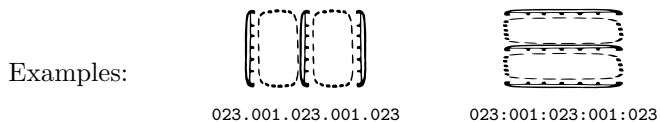


However, there are many cases when this is not so, for example, affix 422:



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.



5). (This rule has no analogue 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 placed both in the *D*-orientation in Case 5.1 and in the *U*-orientation in Case

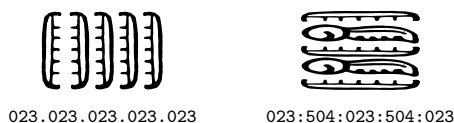
5.2. The combination of these two affixes is declared a central element and the rule 3 is applied.



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:



Note that `\maya{023.(023.023).023.023}` gives  $\left(\left(\left(\right)\right)\right)$  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). 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 §6.5 for more detail about local and global action of commands.

The font ‘`codex`’ is already loaded and is current from the very beginning (i.e., from the command `\input mayaps`). It is associated to the control sequence `\codex`.

**Example.** Suppose that you have Maya font files `fnt1.mpf` and `fnt2.mpf`. Suppose that no font changes were done before (thus, the current font is ‘`codex`’). Then the commands

```
\mayaFont\one=fnt1
\mayaFont\two=fnt2
\maya{422} \one \maya{422} \maya{422} \codex \maya{422}
{ \two \maya{422} \maya{422} } \maya{422}
```

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. The 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 §3.3, rule 2, and §3.4). The default value (when the argument is omitted) is  $[(\text{r})(\text{l})(\text{R})(\text{r})]$ .
- *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)(\text{r})(\text{R})(\text{r})(\text{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  $L$  (respectively,  $R$ ) orientation is obtained from it by turning the image from the file clockwise (respectively, counterclockwise).

**Example 3.** Assume that the file `empty.mpf` is the empty Maya font (see Example 1 in §9.5). 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 the backslash characters `\`. Usually such characters appear in `eps` files when bitmaps (rasterized images) are included there, but not only in this case. I checked that `eps` files produced by `xfig` are good for `\mayaAddGlyph` if bitmaps are not inserted into them. The files produced by `autotrace` and `cotrace` are also good.




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 §11). The font ‘`codex`’ used here is prepared with it. If you cannot install `cotrace` on your platform or if you are not satisfied by `cotrace`’ quality, try another one (I recommend to try `autotrace` but there is a lot of other available vectorizers).

**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`).


## 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 (T<sub>E</sub>X, 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{560.113}`, you get  instead of . The ligature mechanism of MayaPS just replaces each occurrence of ‘`560.113`’ in glyph codes with ‘`561`’ before interpreting them. So, if you type directly `\maya{561}`, you obtain the same result .







If you want to get  instead of , you may type `\maya{(560).113}`. There is also a more permanent solution: the command

```
\mayaDeleteLigature{560.113}
```

cancels this ligature. Note that `\maya{{560}.113}` 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{560.113}` discussed above has the following property. If we cancel it by the command `\mayaDeleteLigature{560.113}`, then the code `560.113` 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{359<023:023>}` produces . Here the composed glyph `023:023 = ` is inserted into `359 = `. The obtained glyph  has its own name `373`, i.e., the commands `\maya{359<023:023>}` and `\maya{373}` are equivalent. Other examples are:

173 =  inside 321 =  gives 321<173> = 327 =  ;  
 204 =  inside 359 =  gives 359<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).** The glyph names in the font ‘codex’ are given according to the catalogue of Maya glyphs composed by Bruno Delprat on the base of the catalogue from the book [1]. However, many specialists in ancient Maya are more familiar with Thompson’s catalogue [7]. The codes from Thompson’s catalogue 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.



Note, that from MayaPS’ point of view there is no difference between ligatures and different encodings. All of them are just substitution rules (see the next subsection) and they are treated by the same algorithms.

**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 and Thompson codes which are discussed above.

In this text we shall denote substitution rules by  $s_1 \rightarrow s_2$  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`:

560.113  $\rightarrow$  561                      321<173>  $\rightarrow$  327              T1  $\rightarrow$  026  
 359<023:023>  $\rightarrow$  373                    359<204>  $\rightarrow$  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{T738.T62}` gives , but not . This means that the substitutions T738  $\rightarrow$  560 and T62  $\rightarrow$  113 are applied but the substitution (ligature) 560.113  $\rightarrow$  561 is not applied to the result.

Substitutions are applied to a string  $w$  by the following algorithm. First, we try to apply substitution rules for the longest possible initial substring. If we succeed, then we proceed to the rest of the string. Otherwise we leave the first character as it is and also proceed to the rest of the string. For example, if the substitution rules

$xy \rightarrow 1, \quad xx \rightarrow 2, \quad yx \rightarrow 3, \quad xxx \rightarrow 4$

are defined for a current font, then `\maya{xyxyxyxyxxxx}` will produce the same result as `\maya{123234}`.

#### 5.4. Definition/canceling of substitutions rules.




A substitution  $s_1 \rightarrow s_2$  for a current Maya font can be defined/canceled by the commands `\mayaDefine{s1}{s2}` and `\mayaUndefine{s1}`.

`\mayaAddLigature{s1}{s2}` and `\mayaDeleteLigature{s1}{s2}` 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 §6.5). 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{s1}{s2}   \mayaGlobalAddLigature{s1}{s2}
\mayaGlobalUndefine{s1}     \mayaGlobalDeleteLigature{s1}
```

**Remark 1.** If you define new substitution rules, be careful about their eventual conflicts with those which are already (pre)defined. Let us illustrate it by the following example. Suppose that Thompson codes were defined with ‘T’ after the number (instead of ‘T’ before the number, as, fortunately, they are defined in the font ‘codex’). Then they might conflict with ligatures. For instance, in this case, `\maya{321.135T}` would give the same result as `\maya{253T}` because of the ligature  $321.135 \rightarrow 253$ . Indeed, the algorithm from §5.3 starts by trying to apply substitutions to the longest initial substring of ‘321.135T’ which is ‘321.135’.

This problem does not appear when ‘T’ precedes the number. For example, despite the ligature  $135.119 \rightarrow 142$ , if you type `\maya{T135.119}`, you get the same as `\maya{015.119}`, because, in this case, ‘T135’ (from  $T135 \rightarrow 015$ ) is the longest initial substring to which a substitution can be applied.




**Remark 2.** We do not recommend to mix encodings in the same glyph code (see the previous remark).

#### 5.5. Red numerals in the font ‘codex’.

In MayaPS it is not possible to change colors in existing fonts. However, when a font is being created, any valid ps program may be used to draw a glyph. In particular, such a program may draw any glyph in any colors. Standard L<sup>A</sup>T<sub>E</sub>X packages for text coloring do not change the colors of Maya glyphs.

Since the red color has a semantic meaning in ancient Maya scripts, we provide some tools to work with it.

There are two types of numerals in the font ‘`codex`’: usual (black) numerals (glyph codes starting with ‘9’) and red numerals (glyph codes starting with ‘8’). There are two versions of red numerals in ‘`codex`’: the colored version and the black-and-white version. In the first one, red numerals are really red. In the second one, they are white with a black outline.

<i>Black:</i>		•	• •	• • •	• • • •	—	etc.
	900	901	902	903	904	905	
<i>Red (B/W):</i>		◦	◦ ◦	◦ ◦ ◦	◦ ◦ ◦ ◦	—	etc.
	800	801	802	803	804	805	
<i>Red (colored):</i>		•	• •	• • •	• • • •	—	etc.
	800r	801r	802r	803r	804r	805r	

The macro `\codexred` switches the current Maya font to `\codex` and declares the substitutions `800 → 800r`, `801 → 801r`, etc.

The macro `\codexbw` switches the current Maya font to `\codex` and cancels the substitutions declared by `\codexred`.

Thus, after `\codexred` all the glyphs `800`, `801`, `802`,... are printed in the colored version even if their codes are typed without ‘`r`’ at the end. After `\codexbw` they are printed again in the black-and-white style. For example,

`\codexred\maya{803:805 001:803} \codexbw\maya{803:805 001:803}`

yields: 

It is convenient to use the macro `\codexred` 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. (without ‘`r`’) for red numerals. If you want to get a colored version of the text, put the line `\codexred` just after `\input mayaps`. If you want to get a black-and-white version, remove this line (or put the comment sign ‘`%`’ before it).

The macros `\codexred` and `\codexbw` discussed above are just abbreviations of `\codex\mayaRed` and `\codex\mayaBW` (the command `\codex` sets the font ‘`codex`’ and then the commands `\mayaRed` or `\mayaBW` define or cancel the substitution rules). The macros `\mayaRed` and `\mayaBW` can be applied to any Maya font based on the encoding used in `codex`. For example, they can be applied to the font ‘`gates`’ (a plumb font, created by William Gates in the 30’s and adopted for MayaPS by Bruno Delprat).

## 5.6. The macro `\mayaDebug`.

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 check if the obtained glyph code is valid. Neither it checks if primitive glyph names are defined. It just applies the substitutions.

**Examples.** `\mayaDebug{359<204>xyz}` yields `367xyz`;

`\mayaDefine{xx}{1}\mayaDefine{xy}{2}\mayaDebug{T1.xxxxxyy}`


yields `026.11yy` (guess, why).

**Restriction.** (Similar to §6.5.) Ampersand character ‘&’ is not allowed in the argument of `\mayaDebug`.

## 6. Glyph showing and paragraph formatting.



### 6.1. The 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 [2; 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 §6.4). 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 [2; Exercise 13.1] for more detail).

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.abc}`). 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 written in the “hidden”  $\TeX$ ’s registers `\maya@xsize` and `\maya@ysize` (they are “hidden” in the sense that a user has no direct access to them unless he changes the `catcode` of ‘@’). The ratio of these parameters is always  $h/w = 15/23$  (recommended by Bruno Delprat) unless you use the command

`\mayaIInsistToChangeWidthAndHeightIndependently{new w}{new h}`

We do not recommend to use it (this is why its name is so long). 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.

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 square  $h \times h$  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     ...

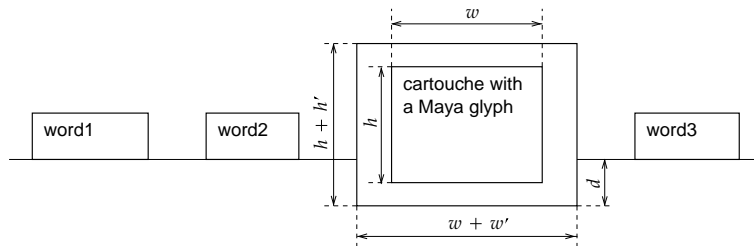
(note, that `abc\mayaGlyph{ }\dots` yeilds `abc`  ...).

## 6.2. The 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 §6.3), 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  $\text{\TeX}$  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  $d = \frac{3}{7}(h - 1.75\text{ex}) + \frac{1}{2}h' - c$  where  $h$  is the cartouche height,  $h'$  is the value of `\mayavspace`,  $c$  is the value of `\mayavcorrection`, and  $\text{ex}$  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.

### 6.3. The 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.,  $(new\ h)/(new\ w) = (old\ h)/(old\ w)$ .

It sets also `\mayahspace` =  $\frac{1}{6}w$ , `\mayavspace` =  $\frac{1}{3}h$ , and `\mayahskip` =  $\frac{1}{15}h$  plus  $\frac{1}{30}h$  minus  $\frac{1}{30}h$ .

### 6.4. The macro `\maya{GlyphCode1 GlyphCode2 ... GlyphCoden}`

Writes a sequence of Maya glyphs separated by spaces (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 (after the last) glyph code are allowed but ignored.

The action of `\maya` is controlled by the parameters from §6.2 and also by the skip (glue) register `\mayahskip`. If `\mayahskip` is set to zero, then `\maya{g1 g2 ... gn}` is equivalent to

`\mayaGlyphInLine{g1}` `\mayaGlyphInLine{g2}` ... `\mayaGlyphInLine{gn}`.

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

The value of `\mayahskip` is set automatically by the `\mayaSize` macro (see §6.3), but it can be redefined in a standard way. For example, by the command `\mayahskip=0pt\relax` or by `\mayahskip=1mm plus 0.5mm minus 0.5mm`

## 6.5. Glyphs with Captions.

### 6.5.1. 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}
```

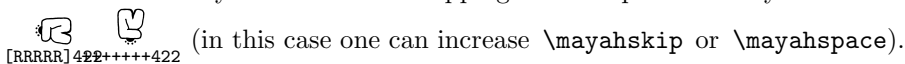
then you obtain:



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 written.

The distance between the bottom of the cartouche and the baseline of the caption is equal to the current distance between the baselines of usual text lines, i.e. the current value of the  $\TeX$ 's parameter `\baselineskip`. Recall that the *baseline* is the line through the bottoms of letters like a, e, x (not like g, q, y).

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. However this may lead to the overlapping of the captions if they are too long:



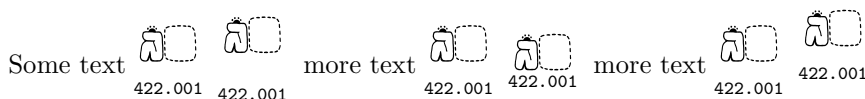
**6.5.2. Restriction.** The argument(s) of the caption-macros cannot contain the character `&`. Fortunately, the `&`-modifier is equivalent to `[c]`. Maybe, some  $\TeX$ erts will explain me how to remove this restriction in future versions (as well as a space in captions before `‘:’`, `‘!’`, `‘?’` when the package `[french]{babel}` is used). To print `‘&’` in the caption, you may use the `*`-version of caption-macros (see §6.5.5).

**6.5.3. Parameters.** The caption-macros are controlled by the same parameters as their no-caption prototypes (see §§6.1 – 6.3) and also by the dimension register `\mayavspaceC`. Its value is added to the distance between the cartouche and the caption text. The default value is zero. It may be positive or negative.

Example:

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

produces



**6.5.4. Font in captions.** By default, the font `cmTT8` scaled at 7pt is used in the captions. The size of this font can be changed by the commands `\mayaCfive`, `\mayaCsix`, ..., `\mayaCsixteen`. For example,

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



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\font=cmr10 at 15pt
\mayaC{?422} { \let\mayaCaptionFont=\font \mayaC{422} }
\mayaC{?422} { \let\mayaCaptionFont=\it \mayaC{?422} }
```

yields 
  
<sub>?422</sub> **422** <sub>?422</sub> *?422*

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

```
\dots\mayaGlyphC*\bf arm}{&314}\mayaGlyphInLineC*{\&314}{&314}
```

yields: ... **arm** 
  
&314

## 6.5. Local and global action of commands.

Recall (see [2] 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 [2; 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 which cause global/local change of MayaPS state.

Global	Local
<code>\mayaGlobalDefine</code>	<code>\mayaDefine</code>
<code>\mayaGlobalUndefine</code>	<code>\mayaUndefine</code>
<code>\mayaFont</code>	<code>\cs</code> (defined by <code>\mayaFont\cs=file</code> )
<code>\mayaAddGlyph</code> (retroactive)	<code>\mayaSize</code>
	<code>\mayaCfive, ..., \mayaCsixteen</code>
<code>\global\mayahskip=glue</code>	<code>\mayahskip=glue</code>
<code>\global\register=dimen</code>	<code>\register=dimen</code>
<code>\global\let\mayaCaptionFont=\cs</code>	<code>\let\mayaCaptionFont=\cs</code>

(here `\register` is one of `\mayahspace`, `\mayavspace`, `\mayavcorrection`, `\mayavspaceC`).

## 7. Error diagnostics.

**7.1. Errors detected by T<sub>E</sub>X.** These are errors appeared, for example, if you misspell a name of a macro, if you forget a brace, etc. The standard T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X mechanism of the error diagnostics works in this case. If the argument of a caption-macro (see §6.5) contains the ampersand ‘&’, then T<sub>E</sub>X detects an error at the moment of printing the caption.

The most disturbing error (known to me) which may occurs when MayaPS is used, is caused by spaces at the end of the arguments of the commands `\mayaGlyph`, `\mayaGlyphInLine`, and their caption versions. In the current version of MayaPS, in this case T<sub>E</sub>X stops with an error message like this

```
! Argument of \maya@sA has an extra }.  
<inserted text>  
      \par  
      . . . . .  
  
1.15 \mayaGlyphInLine{ }  
?  
Runaway argument?
```

If you press ‘Enter’, you get another message as strange as this one. It is not clear from these messages 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 these messages (by keeping to press ‘Enter’, or by replying `q` to T<sub>E</sub>X), then you have a good chance to compile correctly the rest of the document.

**7.2. Errors detected by a PostScript interpreter** (i.e., error messages appeared while printing or viewing the `ps` file). 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). Of course, it is not a very difficult exercise to find an argument of `\mayaGlyph` which yields a corrupted `ps` file. However, a solution of this exercise is too complicated to be written by an occasional mistake in a glyph code. Thus, if T<sub>E</sub>X/MayaPS/Dvips produce a `ps` file (and if you don’t use bad `mpf` or `eps`), then it should be a valid PostScript file despite all error messages of T<sub>E</sub>X. If this is not so, then it is a bug and I will be grateful if you inform me about it saying also which version of Dvips was used (I don’t pay for bugs as Knuth does).

**7.3. Errors in glyph codes.** See §6.1.

**7.4. 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 T<sub>E</sub>X writes his messages. Usually, this is the file `.log` and/or the terminal.

## 8. How MayaPS works (interaction T<sub>E</sub>X/Dvips).

In this and in the next sections (§8 and §9) we assume that the reader knows something about PostScript (see [8] for a short introduction and [4] for a complete description of the PostScript language). Understanding of this section is not necessary for usage of MayaPS, with existing fonts, but it may be helpful for creating new fonts.

### 8.1. 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 T<sub>E</sub>X command `\special{header=file}` makes Dvips to include a file into the prolog of the resulting `ps` file.

The T<sub>E</sub>X 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 T<sub>E</sub>X 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 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.

**8.2. 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
%@ g1
Description of g1
%@ g2
Description of g2
...
%@
end end end end
```

where  $g_1, g_2, \dots$  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 §9.

### 8.3. How MayaPS works. The standard mode.

If everything runs well, then the flag `\ifmaya@TmpFile` is switched on and 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 §8.2) 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  $g_{i_1}, g_{i_2}, \dots$  (for example,  $g_5, g_{13}, \dots$ ) of a font  $F$ , glyphs  $g'_{j_1}, g'_{j_2}, \dots$  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 §10.3
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}`.

#### 8.4. The mode `\maya@TmpFilefalse`

If the flag `\ifmaya@TmpFile` is switched off (this can be done only by changing the file ‘`mayaps.tex`’ or by changing the `catcode` of `@`), then all Maya fonts are just included to the prolog of the resulting `ps` file by the commands `\special{header=..}`.

Similarly, if `mayaps.pro` or an `mpf` file is not found by `TeX`, then `TeX` issues `\special{header=..}` command 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 ligatures defined in the font file are not loaded.

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

## 9. 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 [8]. In the next two sections we explain how to create a Maya font out of `eps` files without any knowledge of PostScript.

**9.1. 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’.

**9.2. General Structure. Comments.** A file `mpf` (MayaPs Font) contains PostScript programs which are (partially) included by Dvips into the resulting `ps` file (the way how they are included is described in §8). It consists of two sections:

- **Font Header** (see §9.3).
- **Glyph Description Section.** (see §9.4).

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

Comments may not start with the combinations `%@` and `%L@` 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.

Empty lines are allowed in `mpf` files. They are ignored.

**Remark.** In §9.3 and §9.4, when we explain the action of PS commands from an `mpf` file, we assume for simplicity that the flag `\ifmaya@TmpFile` is switched off (see §8.4).

**9.3. 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 §9.1), 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 any number of substitution (ligature) declarations. These are lines of the form

```
%L@ s1 s2
```

(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{s1}{s2}` described in §5.3. Substitution declaration lines may be placed everywhere before the end-of-header line ‘%@’.

#### 9.4. 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):

- `wname` (number) width of glyph;
- `hname` (number) height of glyph;
- `aname` (boolean) true if affix;
- `Aname` (array; optional) [ $(L)$   $(U)$   $(R)$   $(D)$   $(S)$ ] where  $L, \dots, S$  are strings of modifiers (see §3). The default value: [ $( )$   $(\mathbf{r})$   $(\mathbf{l})$   $(\mathbf{R})$   $( )$ ].
- `mname` the procedure of glyph drawing. It should have the form `{GIni PostScript commands GEnd}`. 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 `mname` will be executed (the command `GIni` opens it). So, the names defined in  $D_2$  may be used everywhere. The procedure `mname` should draw the

glyph in the bounding box [0 0  $w$   $h$ ] where  $w$  and  $h$  are the values of `wname` and `hname`.

**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 `wname`, `hname`, `aname`, etc. This means, for example, that a definition `'aname false def'` can be abbreviated up to `'aname 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 the font `'codex'`). 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` command.

## 9.5. 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/gsave/grestoreD
/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

```

and a tex file contains

```

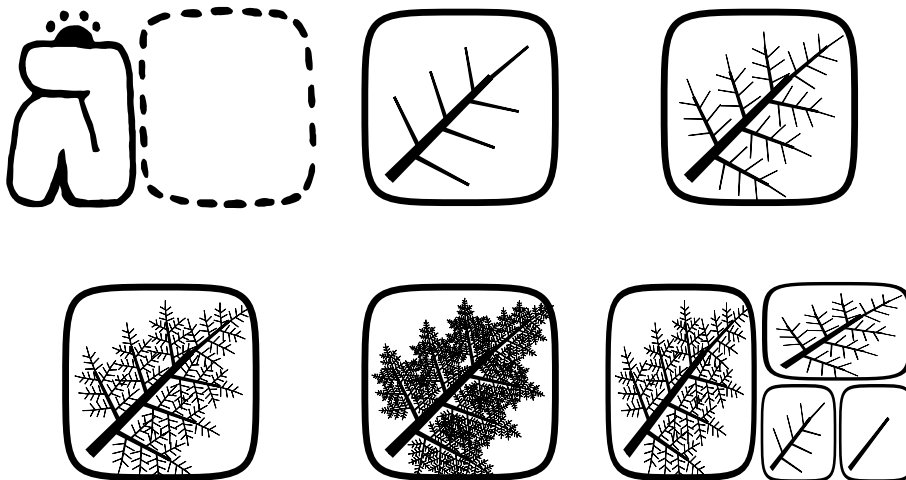
\input maya
\mayaFont\fractal=fractal
\mayaSize{27mm}
\mayahskip=0pt\relax\mayahspace=-3mm
Fonts {\tt codex} and {\tt fractal}:

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

```

Then plain T<sub>E</sub>X (not L<sup>A</sup>T<sub>E</sub>X) and Dvips give the output:

Fonts codex and fractal:



**Exercise.** What is the Hausdorff dimension of the glyph  $\backslash\maya{\infty}$ ? :-)

## 10. 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 §10.2.

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 (see an example in §11).

### 10.1. 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).

### 10.2. 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 box which contains 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 §6.3) 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
```

Moreover, the most of EPS files produced by standard programs (graphical editors, vectorizers etc.) 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
```

### 10.3. The simplest (not the best) way to make an MPF.

An MPF file may be composed as follows (see in §9.2 about comments and §9.3 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 §9.1; for example,  $(s)$  is (89) in `codex.mpf`) and  $k$  may be set to  $5 \times$  (the number of glyphs) + 1 (see §9.4 for more detail). 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,  $L, \dots, S$  are the orientations as in §3.3 (the default value is [(O)(lR)(l)(R)(O)]), and  $[llx, lly, urx, ury]$  is the boundary box of the EPS file (see §10.2).

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).

**10.4. 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@ s11 s12
```

```
%L@ s21 s22
```

```
...
```

```
%@
```

```
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 §10.3. Each Glyph Description has the following structure:

```
%@ name
```

```
/wname w def /hname h def /aname 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 §10.3.

### 10.5. MPF file created using cotrace.

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@ s11 s12
%L@ s21 s22
...
%@
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 files mpf created as described in §11 have this structure.

### 10.6. 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  $P_1, P_2, \dots, P_n$ . In this case the MPF file described in §10.4 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 P1 without DSC Header Comments
}bind def
/prolog2{save userdict begin
    Prolog P2 without DSC Header Comments
}bind def
...
/prologn{save userdict begin
    Prolog Pn without DSC Header Comments
}bind def end end end
```

2). ‘prolog’ should be replaced by ‘prolog $i$ ’ in each definition of  $mname$ . 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.

## 11. Vectorizer CoTrace.

**11.1. 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`.

### 11.2. Creating a MayaPS font using cotrace and makefont.

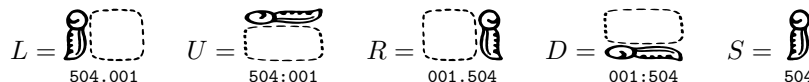
A shell script `makefont` was used for creating the font ‘codex.mpf’. It can be easily adopted for creating other fonts. It runs under UNIX/LINUX and MacOS (from UNIX terminal).


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  $3527 \times 3527$

pixels (the restriction imposed by `cotrace`). The linear sizes of bitmaps used in the font ‘`codex`’ are about 300–600 pixels (for example, ‘`w021 586 D/h021 774 D`’ in ‘`codex.mpf`’ means that the glyph 021 was produced from a bitmap of size  $293 \times 387 = \frac{586}{2} \times \frac{774}{2}$  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



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.

Red versions of red glyphs are obtained also from **black** and white bitmaps (for example, the same BMP files are used for black numerals and colored versions of red numerals in the fonts ‘`codex`’ and ‘`gates`’).

3). Create directory (say, `makefn` where `fn` is the name of the Maya font you are going to create) and its subdirectory `BMP`. put all your BMP files with the glyph images into the directory `BMP`. Compile the C programs and make the shell scripts executable by the commands

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

Copy the files `cotrace`, `addg`, `addr`, `addglyph`, `addredglyph`, and `makefont` (all without extensions) to the directory `makefn`. Modify the file `makefont` which is more or less self-explaining, and run

```
./makefont
```

## 12. Ideas for future versions of MayaPS.

Here I describe some projects of new features of MayaPS which are not immediate to implement. Moreover, I am not sure that they are really needed for users. So, maybe I will implement (some of) them, but only if I know that somebody needs them.

### 12.1. The macro `\mayaPickGlyph{g1}from{font}{g2}`

( $g_1$  and  $g_2$  are glyph names). A macro which defines the glyph  $g_1$  in the current Maya font by taking it from the Maya font file `font.mpf` where it appears under the name  $g_2$ .

### 12.2. The macro `\mayaFontMap{font}`

A macro which reads the Maya font file `font.mpf` and shows each glyph of this font with its name and with the default orientations for affixes.

### 12.3. Selection mechanism for `\mayaAddGlyph`

MayaPS includes only those glyphs from `mpf` files which are really used in the document. However, if a glyph is defined by the macro `\mayaAddGlyph`, then the `eps` file is included into the resulting `ps` file even if the glyph is never used in the document. Maybe, it would be helpful to modify the macro `\mayaAddGlyph` so that it includes only glyphs which are really used (see Remark 2 in §10.3).

### 12.4. New mechanism for different encodings.

The possibility to use several encodings for the same font is realized via substitutions (see §5.2). This may cause problems indicated in Remark 1 in §5.4. I see two reasonable ways to avoid (some of) these problems.

1). To change the mechanism of substitutions so that a substitution rule is applied to a substring  $s_1$  of a string  $s$  **only** if the initial character of  $s_1$  is either ‘(’, or ‘<’, or it is the first character of a glyph name and this name cannot be extended to the left (similar claim for the last character of  $s_1$ ).

2). To keep the general substitution mechanism as it is, but to introduce a special case of substitutions (encodings) which can be applied only to the whole glyph names (primitive glyph codes) and not to parts of them, nor to complicated expressions.

### 12.5. Type 1 fonts.

If primitive glyphs were realized via type 1 fonts (see [4]), then the `pdf` file obtained from `ps` could be much shorter.

For example (for the current version of MayaPS), if a `tex` file contains 15000 times the glyph `\maya{353}` and nothing else, then the file sizes are 1Mb for `.ps`, 30Kb for `.ps.gz`, but after a conversion to `pdf` using `GSview/gs854w32-gpl` the size increases up to 100Mb. The convertor `epstopdf` from the package `tetex3` gives similar results.

A more realistic example: the file sizes for this manual are: `.ps.gz`(200K), `.ps`(500K), and `.pdf`(900K).

## References

- [1]. E.V. Evreinov, Yu.G. Kosarev, B.A. Ustinov, *Usage of Electronic Computational Machines for Study of the Ancient Maya Written Language*. Siberian Branch of the Acad. Sci. USSR, Novosibirsk, 1961, (in Russian).
- [2]. D.E. Knuth. *The T<sub>E</sub>Xbook*. Addison-Wesley, Boston, 1984.
- [3]. L. Lamport. *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System* (2nd ed.), Addison-Wesley, Boston, 1994.
- [4]. *PostScript Language Reference Manual*. Files `plrm.pdf`, `plrm2.pdf` available on <http://www.adobe.com>
- [5]. T. Rokicki. *Dvips: A DVI-to-PostScript Translator*. The file `dvips.pdf` included in the most of T<sub>E</sub>X distributions; available on <http://www.ctan.org>
- [6]. A. Syropoulos. *Typesetting Native American Languages*. Journal of Electronic Publishing, **8**(2002), issue 1. <http://www.press.umich.edu/jep>
- [7]. J.E.S. Thompson, *A Catalog of Maya Hieroglyphs*, Univ. Oklahoma Press, 1962
- [8]. 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`