Xdvipsk: Dvips ready for more image formats and accessible publications

Sigitas Tolušis
Mindaugas Piešina
Valentinas Kriaučiukas
www.vtex.lt

2026 May 2

EuroBachoTeX 2026

Introduction
Preliminaries
xdvipsk-support
Unicode in PostScript
Unicode for Type 1 fonts
Selecting Unicode values
Glyph renaming
Lua callbacks
Final notes

Introduction

Motivation and history

- The workflow TEX → DVI → PS → PDF and tools around are in **our production**
- Still a lot of type1 Postscript fonts
- Stable production requires renovation of old tools
- `Dvips` shortcomings:
  - restricted support of graphic formats (extended around 2012)
  - missing support of OpenType (OT) fonts (extended around 2016)
  - missing support of Unicode maps (extended in 2025, included in TEXLive)

Was given: about year 2016

- **x** denotes the Unicode (OT fonts) extension
- **k** denotes use of the Kpathsea library (as with `dvipsk`)

Announced in public: S. Tolušis, A. Povilaitis, V. Kriaučiukas. Xdvipsk: Dvips ready for OpenType fonts and more image formats. TUGboat 38(2):197–201, 2017. `https://tug.org/TUGboat/tb38-2/tb119tolusis.pdf`

- described graphics extension
- described the initial variant with OT fonts

The boundary conditions

`xdvipsk` is the middle of workflow
TEX → DVI → PS → PDF

The step TEX → DVI conditions:

- performed by LuaTEX (`dvilualatex`)
- package `xdvipsk-support` is needed for:
  - the graphics extension
  - the use of OpenType fonts

The step DVI → PS conditions:

- Unicode maps for Type 1 Postscript fonts (potentially useful for any TEX workflow)

The step PS → PDF condition:

- performed by Adobe Acrobat Distiller

About `pdfwrite` of Ghostscript

1. Unicode mappings in PostScript are `GlyphNames2Unicode` dictionaries
   - `pdfwrite` supports only single UTF-16 code values
   - VTeX is preparing a patch for multiple UTF-16 codes, including surrogate pairs, and UTF-32 codes.
2. `pdfwrite` does not construct in PDF the structure tree, the necessary component for PDF accessibility.
   - VTeX is open for collaboration with Ghostscript developers

Preliminaries

Two kinds of maps

1. **Maps between font names** (`psfonts.map`):

   source: font names in DVI files

   - for example, .tfm file names

   target: font file names

   - .pfb or .otf file names

2. **Maps to Unicode**:

   source: font glyphs

   - glyph names
   - OTF glyph indices (GID)s

   target: Unicode values

   - UTF-16 codes or some UTF-32 codes
   - lists of these codes

Private Use Area (PUA)
`https://en.wikipedia.org/wiki/Private_Use_Areas`
consists of three ranges:

- one in the Basic Multilingual Plane, `U+E000`–`U+F8FF`,
- in plane 15, `U+F0000`–`U+FFFFD`,
- in plane 16, `U+100000`–`U+10FFFD`.

Intentionally left undefined in the Unicode Standard.
We avoid them as much as we can.

DVI specials

The noun ‘special’ denotes TEX command `\special`.

The terminological economy: mapline special means:

the `\special` command with an argument prefixed with ‘`mapline`’

Package xdvipsk-support

Calling xdvipsk-support

`\RequirePackage[`⟨*option*⟩`]{xdvipsk-support}`
`\documentclass...`

- Loads `xdvipskmaps.sty` (optionally),
- loads `xdvipskmaps.lua` (optionally),
- loads `xdvipsk.def`,
- replaces default `luaotfload` DVI driver `dvisvgm`, if started, with `xdvipsk`.

The proper way is not to start `dvisvgm`: set in the local `luaotfload` configuration file `luaotfload.conf`:

```
default-dvi-driver = xdvipsk
```

It would be nicest to have an option of `luaotfload` for this.

One of the things `xdvipsk-support` does: enables `xdvipsk` bitmap graphics extension:

- `xdvipsk` accepts images in formats BMP, JPEG, PCX, PNG, and TIFF
- all formats should be declared in the driver file ⟨*driver*⟩`.def`, most likely `xdvipsk.def`:

```
\@namedef{Gin@rule@.tif}#1{{bmp}{.tif.bb}{#1}}
\@namedef{Gin@rule@.tiff}#1{{bmp}{.tiff.bb}{#1}}
\@namedef{Gin@rule@.jpeg}#1{{bmp}{.jpeg.bb}{#1}}
\@namedef{Gin@rule@.jpg}#1{{bmp}{.jpg.bb}{#1}}
\@namedef{Gin@rule@.png}#1{{bmp}{.png.bb}{#1}}
```

  where ⟨*drive*⟩ name should be used in TEX document preambles like `\usepackage[`⟨*driver*⟩`]{graphics}`

OpenType extension

Another thing `xdvipsk-support` does: prepares OTF-related maps for `xdvipsk`.
Package options all are related with this extension:

`noOpenType`: disables OT font support (does not load `xdvipskmaps.sty` and `xdvipskmaps.lua`)

`otfchar`: (default)

- sets to use TEX charcodes for indexing glyphs
- creates folder `.xdvipsk`
- writes there files map for OT fonts
- writes there Unicode maps for OT fonts

`otfgid`: (not possible for LuaTEX binaries up to v.1.25.7)

- sets to use `GID` encoding for OT fonts
- sets to write file map lines directly into the DVI file as `mapline` specials

Use of `.xdvipsk` folder

- the initial implementation of OT font support (described in our earlier paper)
- file ⟨*article*⟩`.opentype.map` of lines

  ⟨*tfm name*⟩→→⟨*ps name*⟩→⟨*texfont name*⟩→>⟨*file name*⟩

  elements separated by tabs ('→')
- files ⟨*psname*⟩`.encodings.map` of lines

  ⟨*charcode*⟩,⟨*GID*⟩,⟨*Unicode*⟩,⟨*glyph width*⟩,⟨*glyph height*⟩

```
59965,708,00AF,376832,414842.88
59966,709,00200331,376832,0
59967,710,0304,0,573440
59968,711,02DA,569507.84,408944.64
59969,712,0020030A0301,569507.84,543948.8
59970,713,0020030A0301,569507.84,586547.2
59971,714,030A,0,610140.16
```

Without .xdvipsk folder

- Lines of ⟨*article*⟩.opentype.map become arguments of mapline specials

  mapline:␣⟨*tfm name*⟩␣⟨*ps name*⟩␣⟨*texfont name*⟩␣ >⟨*file name*⟩
- OT fonts in DVI are encoded by GIDs instead of TEXcharcodes
- To inform xdvipsk about this options, the command

  \special{vtex:settings.xdvipsk.opentype={enc=gid}}

  is inserted in the DVI file.

Unicode in Postscript files

Unicode for Type 1 PostScript fonts

- There is an undocumented dictionary
- Keys are glyph names
- Values are strings of hexadecimal digits
- It is stored in `FontInfo` dictionary
- under key `GlyphNames2Unicode`
- Distillers (Acrobat and Ghostscript) use it

An excerpt from this dictionary for MSAM10 font:

```
/GlyphNames2Unicode <<
  /squaredot <22A1>
  /squareplus <229E>
  /squaremultiply <22A0>
  /square <25A1>
  ...
  >> def
```

Constraints on hexa-strings to be Unicode values

1. The string length is a multiple of 4
2. The quadruples are UTF-16 codes
3. Two consecutive codes can be a *surrogate pair*
   `https://en.wikipedia.org/wiki/UTF-16`
   encoding an UTF-32 code
4. Surrogate values occur in surrogate pairs only

```
/GlyphNames2Unicode <<
    /FFIsmall <006600660069>
    /Bmathdstruck <D835DD39>
    >> def
```

Unicode for OpenType fonts in PostScript file

Use Opentype glyphs indexes (GIDs) instead of glyph names in `GlyphNames2Unicode` dictionaries:

```
/GlyphNames2Unicode <<
    5 <0041>
    6 <0042>
    7 <0043>
    ...
    517 <00E9>
    582 <00ED>
    640 <0142>
    >> def
```

Unicode maps for Type 1 PostScript fonts

G2u tables

- `xdvipsk` still reads if Kpathsea finds ⟨*fontname*⟩`.g2u` maps (but deprecated)
- their syntax is as of `GlyphNames2Unicode` dictionaries

One more example from the `.ps` file of this presentation for font `LMRoman10`:

```
/GlyphNames2Unicode <<
  /angleleft <2329>
  /angleright <232A>
>> def
```

We needed more possibilities and more convenient syntax for UTF-32 codes

Lua table format for Unicode maps

Example of a simple Lua table (for MSAM10 font):

```
return {
  ['arrowaxisleft'] = { 0xFFE9 },-- HALFWIDTH LEFTWARDS ARROW
  ['arrowaxisright'] = { 0xFFEB },-- HALFWIDTH RIGHTWARDS ARROW
  ['circleequal'] = { 0x2257 },-- RING EQUAL TO
  ...
  ['muchgreater'] = { 0x22D9 },-- VERY MUCH GREATER-THAN
  ['muchless'] = { 0x22D8 },-- VERY MUCH LESS-THAN
  ['squaresmallsolid'] = { 0x2B1D },-- BLACK VERY SMALL SQUARE
  ['star'] = { 0x2605 },-- BLACK STAR
}
```

- Glyph names as table keys are double or single quoted
- Optional comments with the Unicode symbol names for convenience

Lua tables

Some lines from the Lua table for MSMB10 font:

```
['notfollowsoreql'] = { 0x2AB0, 0x0338 },
['lessornotsimilar'] = { 0x22E6 },
['greaterornotsimilar'] = { 0x22E7 },
['A'] = { 0x1D538 },
['B'] = { 0x1D539 },
['C'] = { 0x2102 },-- DOUBLE-STRUCK CAPITAL C
```

- Multiple codes separated by commas and optional spaces
- Direct use of UTF-32 codes

A string for glyph renaming (for t1xttsc font):

```
['scaron'] = { 0xA731, 0x030C, 'ScaronSC' },
-- LATIN LETTER SMALL CAPITAL S/COMBINING CARON
```

Adobe list sets ["scaron"]={0x0161}

Unicode maps in TEX Live

All in `texmf-dist` subtree:

Adobe: `fonts/map/glyphlist/glyphlist.txt`,
pdftex: `tex/generic/pdftex/glyphtounicode.tex`
luaotf:
`tex/luatex/luaotfload/luaotfload-glyphlist.lua`,

ntx: `tex/latex/pdfx/glyphstounicode-ntx.tex`,
pdf: `fonts/map/glyphlist/pdfglyphlist.txt`,
tex: `fonts/map/glyphlist/texglyphlist.txt`,
cmr: `tex/latex/pdfx/glyphstounicode-cmr.tex`,
cmex:
`tex/latex/latex-lab/glyphstounicode-cmex.tex`,
cs: `tex/csplain/fonts/glyphstounicode-cs.tex`,

The builtin `xdvipsk` Lua table `glyphlist_table.lua` is constructed from them

Some statistics:

Map name	Number of glyphs	Not in a private area
Adobe	4281	4089
pdftex	5505	5450
luaotf	4291	4099
ntx	383	374
pdf	373	344
tex	324	309
cmr	277	276
cmex	132	132
cs	4	4
builtin	4683	4683

The algorithm to build `glyphlist_table.lua`

1. Start with the **Adobe** map as it is
2. Add glyph names (their maps) not yet included from **pdftex**, except the ones prefixed with `tfm:`
3. Add glyph names not yet included from **luaotf**.
4. Add all glyph names from **cs**.
5. Add glyph names not yet included from **cmr**, except a$\langle num \rangle$ or d$\langle num \rangle$, where $\langle num \rangle$ is an integer.
6. Add 6 glyph name maps from **tex**:
   - `altselector` $\mapsto$ `0xD802`,
   - `ascendercompwordmark` $\mapsto$ `0xD80A`,
   - `capitalcompwordmark` $\mapsto$ `0xD809`,
   - `emptyslot` $\mapsto$ `0xD801`,
   - `ringfitted` $\mapsto$ `0xD80D`,
   - `twelveudash` $\mapsto$ `0xD80C`.

Logic of unification

- **Adobe** "as it is": to know when to rename glyph name.
- Maps **pdftex** and **luaotf** provide distinct values for some Hebraic/Arabic glyphs.
- Glyph names, prefixed with 'tfm:' in **pdftex**, go to individual font maps.
- Include **cs** earlier than **cmr**.
- Glyph names a$\langle num \rangle$ and d$\langle num \rangle$ go to individual font maps.
- **cmr** subsumes the **cmex**.
- **pdf** is a subset of **Adobe**.
- **tex** is kind of a multivalued map and requires a special treatment.

Internal xdvipsk maps

The list of (22) directories in TEX tree with mapped fonts:

```
amsfonts/cm
amsfonts/cyrillic
amsfonts/euler
amsfonts/latxfont
amsfonts/symbols
hoekwater/manfnt
public/cjhebrew
public/lm
public/marvosym
public/mathpazo
public/mlmodern
public/mnsymbol
public/newtx
public/niceframe
public/old-arrows
public/prodint
public/pxfonts
public/stmaryrd
public/txfonts
public/wasy
public/xypic
public/yhmath
```

Extension ways for users

- To use external tables ⟨*fontname*⟩`.lua`
- to use `g2umapfile` specials to load the file with whatever name
  - to use one Lua table for glyph names from different fonts, just prefix conflicting keys with `pfb:`⟨*fontname*⟩`/`
- To use `g2umapline` specials to define Unicode values for single glyph names.

Without regenerating `psfponts.map`, the user can introduce new maps between font names using `mapfile` and `mapline` specials:

```
\special{mapfile: myriadd.map}
\special{mapline: ptmbi8r Times-BoldItalic
    " .167 SlantFont " <8r.enc <ptmbi8a.pfb
    " TeXBase1Encoding ReEncodeFont "}
```

How `xdvipsk` selects Unicode values

Sources for construction `GlyphNames2Unicode`

Case for a Type 1 PostScript font ⟨*fname*⟩`.pfb`:

- The builtin Lua table `glyphlist_table.lua`
- The internal joined table `font_glyphs_maps.lua`
- File ⟨*fname*⟩`.lua`, if Kpathsea found such
- Lua tables from the `g2umapfile` specials

  `\special{g2umapfile=`⟨*filename*⟩`.lua}`
  `\special{g2umapfile=pfb:`⟨*fname*⟩`/`⟨*filename*⟩`.lua}`
- Single-glyph maps from `g2umapline` specials

  `\special{g2umapline=`⟨*glname*⟩`,`⟨*uval*⟩`}`
  `\special{g2umapline=pfb:`⟨*fname*⟩`/`⟨*glname*⟩`,`⟨*uval*⟩`}`

  ⟨*uval*⟩ is a space-separated list of UTF-16 or UTF-32 codes

Rules for selecting values between maps

The descendant priority order of map selection rules for a font ⟨*fname*⟩ and a glyph ⟨*gname*⟩:

1. `\special{g2umapline=pfb:`⟨*fname*⟩`/`⟨*gname*⟩`,`⟨*uval*⟩`}`: the value ⟨*uval*⟩;
2. `\special{g2umapfile=pfb:`⟨*fname*⟩`/`⟨*filename*⟩`.lua}`: the value of key ⟨*gname*⟩ in ⟨*filename*⟩`.lua`
3. `\special{g2umapfile=`⟨*filename*⟩`.lua}`: the value of key `pfb:`⟨*fname*⟩`/`⟨*gname*⟩ in ⟨*filename*⟩`.lua`;
4. ⟨*fname*⟩`.lua`: the value of key ⟨*gname*⟩;
5. a table for font ⟨*fname*⟩ in `font_glyphs_maps.lua`: the value of key ⟨*gname*⟩;
6. `\special{g2umapfile=`⟨*filename*⟩`.lua}`: the value of ⟨*gname*⟩ in ⟨*filename*⟩`.lua`;
7. the built-in table: the value of key ⟨*gname*⟩.

Structured glyph names

Adobe glyph list specification. Adobe Systems Inc., 2019
`https://github.com/adobe-type-tools/agl-specification`

- The underscore ('_') in $\langle gname \rangle$:
  - split $\langle gname \rangle$ in '_';
  - search Unicode values for the components;
  - append the Unicode values in the order of the components.
- The dot ('.') in $\langle gname \rangle$:
  - take prefix until the first dot;
  - find a Unicode value for it;
  - return this Unicode value for $\langle gname \rangle$.
- $\langle gname \rangle$ is a ligature like 'fi', 'fl', 'ff', 'ffi' or 'ffl':
  - split to constituent letters;
  - append their Unicode values.

Glyph names with Unicode

- ⟨*gname*⟩ starts with 'uni' & the rest is splittable into UTF-16 codes:
  - this list of UTF-16 codes is the Unicode value for ⟨*gname*⟩, if no surrogate pair occurs in it;
  - otherwise:
    - replace each surrogate pair by the UTF-32 value encoded by it;
    - the resulting list is the Unicode value for ⟨*gname*⟩.
- ⟨*gname*⟩ starts with 'u' & the rest is a single UTF-16 or UTF-32 code: it is the Unicode value for ⟨*gname*⟩.

Examples:

```
uniEB61   -->   0xEB61
uni006600660069   -->   0x66, 0x66, 0x69
uniD835DD39   -->   0x1D539
u1D53A  -->   0x1D53A
```

Glyph renaming

Automatic renaming is done if:

- glyph name is in AGL
- its Unicode value selected for `GlyphNames2Unicode` differs from the one in AGL

The new name for the Unicode value in the form of

- a list of UTF-16 codes: concatenation of 'uni' and the codes;
- a UTF-32 code: concatenation of 'u' and the code:

```
['A'] = { 0x1D4D0 }  -->  /u1D4D0 <D835DCD0>
```

(mathematical bold script letter)

Manual renaming comes from lines of Lua tables, like

```
['A'] = { 0x1D4D0, 'MathBoldScriptA' }
   -->   /MathBoldScriptA <D835DCD0>
```

Problem with renaming in shared encodings

- Encodings are glyph name lists for Type 1 fonts
- so, glyph name change must go in encodings, too

Two lines from `lm.map` with shared encoding `lm-mathsy`:

```
lmsy10 LMMathSymbols10-Regular " enclmmathsy ReEncodeFont "
                                <lm-mathsy.enc <lmsy10.pfb
lmbsy10 LMMathSymbols10-Bold " enclmmathsy ReEncodeFont "
                               <lm-mathsy.enc <lmbsy10.pfb
```

Different Unicode values for the same glyph name:

```
['A'] = { 0x1D49C },-- MATHEMATICAL SCRIPT CAPITAL A
```

for font cmsy10, while for cmbsy10

```
['A'] = { 0x1D4D0 },-- MATHEMATICAL BOLD SCRIPT CAPITAL A
```

A dynamic solution with automatic duplication and optional renaming of conflicting encodings was implemented.

Sources of Unicode maps for OT fonts

Keys in `GlyphNames2Unicode` are GIDs
Sources of GIDs and of Unicode values depend on an option of package `xdvipsk-support`:

- `[otfchar]`

  GIDs: tables ⟨*psname*⟩`.encodings.map` located in subfolder `.xdvipsk`
  Unicode values: ⟨*psname*⟩`.encodings.map`

- `[otfgid]`

  GIDs: the DVI file itself, it is GID-encoded
  Unicode values: OT font files

For glyphs with names, name rules of Type 1 fonts apply.

We did not yet investigate OT fonts on needs to correct Unicode values.

Lua callbacks

Lua callbacks for specials

Effects of results returned by special-related callbacks:

- a non-empty string: a modified DVI special to process;
- `true` value: process the original DVI special;
- empty string or `false` value: skip the DVI special.

Some experimental ideas how to exploit the callbacks:

- to produce different PostScript files from the same DVI file:
  - different versions, like web and print,
  - files with different graphics variants,
- insert PDF graphics into the PDF file by using the coordinate information from this stage;
- compare different versions of the same DVI using information about characters and rules.

List of callback functions

The lua script file `xdvipsk.lua` can define the following callback functions:

- `prescan_specials_callback(`⟨*specar*⟩`,`⟨*lua table*⟩`)` where ⟨*specar*⟩ is the argument of `\special`, and ⟨*lua table*⟩ = `{ hh=`⟨*horizontal pixel position*⟩`, vv=`⟨*vertical pixel position*⟩`, pagenum=`⟨*page number*⟩`, }`
- `scan_specials_callback(`⟨*specar*⟩`,`⟨*lua table*⟩`)`, the arguments are the same as in the previous point
- `after_prescan_callback(`⟨*lua table*⟩`)`
- `after_drawchar_callback(`⟨*lua table*⟩`)`
- `after_drawrule_callback(`⟨*lua table*⟩`)`
- `process_stack_callback(`⟨*lua table*⟩`)`
- `dvips_exit_callback(`⟨*lua table*⟩`)`

After char callback table

{ charcode=⟨*character code*⟩,
  cid=⟨*character index*⟩,
  glyphname=⟨*glyph name*⟩,
  pixelwidth=⟨*character width*⟩,
  rhh=⟨*horizontal pixel position*⟩,
  rvv=⟨*vertical pixel position*⟩,
  bpos=⟨*dvi position begin pointer*⟩,
  epos=⟨*dvi position end pointer*⟩,
  dir=⟨*direction*⟩,
  lastfont=⟨*last used font*⟩,
  psname=⟨*font PSname*⟩,
  tfmname=⟨*font TeXname*⟩,
  fntfile=⟨*font file path*⟩,
  tounicode=⟨*table*⟩,
  pagenum=⟨*page number*⟩, }

Final notes

Storages and development

- `xdvipsk` incorporated into the TEX Live build ecosystem
- we build and test on three architectures: Windows, Linux, and macOS
- the development code is available at `https://github.com/vtex-soft/xdvipsk.git`
- all changes in common with `dvips` files are tagged with C preprocessor macro definition XDVIPSK and conditional inclusions
- The previous version is now frozen and still available at `https://github.com/vtex-soft/texlive.xdvipsk.git`

Thank you!