

Frida

Graphics postprocessing

Frida version 2.3.3c

Incomplete manual preview (version $\gamma 2$) of May 23, 2016

Joachim Wuttke

Scientific Computing Group
Jülich Centre for Neutron Science
at Heinz Maier-Leibnitz Zentrum Garching
Forschungszentrum Jülich GmbH

Homepage: <http://apps.jcns.fz-juelich.de/frida>

Copyright: Forschungszentrum Jülich GmbH 2016–2016

Licenses: Software: GNU General Public License version 3 or higher
Documentation: Creative Commons CC-BY-SA

Author : Joachim Wuttke
Scientific Computing Group
at Heinz Maier-Leibnitz Zentrum (MLZ) Garching

Disclaimer: Software and documentation are work in progress.
We cannot guarantee correctness and accuracy.
If in doubt, contact us for assistance or scientific collaboration.

Contents

1	Introduction	4
2	Graphic formats, workflow	4
2.1	File formats	4
2.2	Workflow for graphics from Frida	5
2.3	Embedding and format conversion	6
2.4	PostScript	7
3	Postprocessing Frida graphics	10
3.1	File structure	10
3.2	Setup	11
3.3	Plot frame and data plot	15
3.4	Extended string format	16
3.5	Text placement commands	19
3.6	Concatenation, insets	20
	References	21
	Index	21

1 Introduction

Frida is a versatile data-analysis program. From a command-line interface, it offers various methods to visualize, manipulate and fit tabular data. Frida is especially designed for the analysis of spectral data, in particular from quasielastic neutron scattering (QENS).

Frida is currently in maintenance mode, which means: bugs will be fixed speedily, new features will be implemented occasionally, but development at large is stalled until a decision is taken which QENS software shall be institutionally supported at MLZ and partner institutes.

The following notes describe how to post-process PostScript graphics files generated by Frida, in order to obtain publication-grade figure. These notes have grown out of an oral tutorial. They will be gradually converted from slide-show format to written-manual style.

In Frida, two different methods are used to visualize data. For screen display in interactive sessions, Frida spawns a Gnuplot session. Plot commands are sent from Frida to Gnuplot through a first-in, first-out pipe. For saving graphics (command `gp`), Frida does *not* use Gnuplot's PostScript backend, but directly writes PostScript commands to a file. In the following, the structure of this Frida-generated PostScript file is explained, and it is shown how to embellish a plot by a few edits to this file.

2 Graphic formats, workflow

2.1 File formats

Graphics can be stored in many different file formats. Table 1 lists formats that are in wide-spread use. One distinction is between *proprietary* and *open* standards. It strongly correlates with the distinction between *platform-specific* and *cross-platform* formats. To facilitate cooperation, we should prefer open, cross-platform standards whenever possible. Platform-specific formats like Windows Metafile (wmf) or Enhanced Metafile (emf) should be avoided.

The most important distinction, however, is between *raster graphics* (bitmaps, pixel maps) and *vector graphics*. In raster graphics, the coloring of each single pixel in an image is specified. Typically, for each of the three base colors, values between 0 and 255 can be specified. Additionally, there may be a fourth number to indicate transparency. So there are three or four bytes per pixel. Accordingly, an image with 1000×1000 pixels (1 Megapixel) is stored in a file of size 3 or 4 MByte.

In contrast, a graphic file in vector format specifies an image in terms of basic plotting operations. Such operations include: choice of color, pen, font; coordinate system transformations; specification of path, clippath, region; drawing lines, arcs, Bezier curves using chosen pen; printing text using characters from chosen font. In this way, complex plots can be stored in a file that takes only a few kBytes.

Photos, and similar images, are most adequately stored as raster graphics. However, technical drawings, and most scientific plots, should normally be stored in vector format. This provides several advantages over bitmaps: Figures can be scaled without quality loss; they can be modified much easier; and they require much less disk space.

ps, eps	vector
svg	vector, XML-based, for WWW
pbm, pgm, ppm, pnm	bitmap (b/w, grey, color, any)
bmp	bitmap, reinvented by MS
png	compressed bitmap, lossless
gif	ditto, was patented, now obsolete
jpg	compressed bitmap, lossy
tiff	container
exif	container, for multimedia
wmf, emf	container, by MS
pdf	ps-based container

Table 1: Frequently used data formats

Therefore, vector graphics, like the plots generated by Frida, should be left in vector format throughout all postprocessing.

2.2 Workflow for graphics from Frida

Within Frida, the commands `p` and `a` are used to plot data and curves. Commands starting with the letter `g` allow to choose output windows, to set data ranges, and to modify other control parameters. Plot commands are sent from Frida to a Gnuplot process, using a first-in, first-out pipe. However, Gnuplot is *not* used for saving graphics to files. Instead, Frida holds a complete copy of the graphics state and of all plotted data in memory. When the command `gp` or `gf` is issued, these data are used to generate a PostScript file that contains essentially the same plot as the active Gnuplot window.

The command `gp` saves a complete graphics file as `l<number>.ps` in the preset graphics output directory (by default `~/gnew/`). For `number`, the lowest unused integer is chosen. The so obtained file can immediately be viewed, printed, and modified in a text editor. Alternatively, the command `gf` generates an incomplete graphic file `l<number>.psa` that is lacking all macro definitions. Such a file is useful when several Frida-generated plots shall be combined into one figure (section 3.6).

I have found it advantageous for my concentration and productivity to do keep data analysis and graphics postprocessing separated. So during a data analysis session, I would generate a number of numbered PostScript files in `~/gnew/`, and take brief notes about graphics that are candidates for later refinement and ultimately for a presentation or/and a publication. Once the analysis is concluded with satisfactory, consistent results, I would leave the Frida session open, narrow down the graphics selection, move the selected figures to an appropriate directory, give them more explicit file names, and then postprocess them. At any moment, I may need to go back to the Frida session to produce new, improved raw figures.

Frida-generated PostScript files consist of human readable structured code in plain ASCII. Any single element in a graphic can be modified by changing the appropriate

Application	Graphics support
LaTeX with dvips	PS
pdfLaTeX	PDF, raster
xeLaTeX	PS, PDF, raster
MS Office	PDF (since 2013), raster
libreoffice	EPS, raster

Table 2: Support for graphic import by text processors and presentation programs. PS means PostScript with bounding box comment; EPS means the same, with file extension `.eps`; “raster” means some raster graphic formats.

section in the PostScript file. To lay a foundation for such modifications, section 2.4 will give a short introduction to PostScript programming, and section 3 will explain structure and content of Frida-generated graphic files. For modifying PostScript files in an interactive session, it is advisable to open a text editor side-by-side with a PostScript viewer. The viewer should update the displayed image as soon as the image file has changed so that a simple “save” command in the editor suffices to see a modified graphic.¹ For debugging, the old-fashioned console program `gs` (ghostscript) is still helpful.

The so obtained embellished Frida graphics should be preserved in PostScript format so that they remain editable. Secondary files in other image formats can be obtained by conversion programs, as discussed in the next subsection.

2.3 Embedding and format conversion

PostScript files, generated by Frida and possibly manually modified, are ready for screen viewing and for printing, but not yet for embedding them in documents or presentations. It is almost always necessary to insert a *bounding box* comment to the PostScript file. Often, the PostScript file must then be converted to PDF. Table 2 gives an overview which embedding applications support which graphic formats.

By default, a PostScript file describes an entire page, with white space all around the graphic. Usually we want to embed just the graphic, not the full page. To accomplish this, we need to insert a *bounding box* comment in our PostScript file. A bounding box is the smallest rectangular box that contains non-white elements of a given graphic. It can be determined with the command²

```
bboxx <filename>.ps
```

When the command is called as

```
bboxx -insert <filename>.ps
```

¹Under Linux, this requirement is met by the open-source PostScript and PDF viewer *Evince*, and probably by several other viewers.

²Under Debian and its derivatives, the program `bboxx` comes in the package `impose+`. Under rpm-based distributions, try the program `bbox` from package `texlive-bbox`.

then the bounding box coordinates are inserted in form of a comment

```
%%BoundingBox: <left> <bottom> <top> <right>
```

near the top of the graphic file. This insertion makes our PostScript file partly compatible with the *Encapsulated PostScript* (EPS) format. An EPS file is a PostScript file with a number of special comments, all starting with `%%`, that mostly contain sectioning information, and thereby help a viewer to navigate between different pages in a multi-page document. Since such sectioning is irrelevant for a single image, the bounding box comment is sufficient to let certain applications accept our file as valid EPS. It may just be necessary to change the file name extension to `.eps`.

Other applications support neither PostScript, nor EPS, but the Portable Document Format (PDF). PDF is a container format that uses lossless compression to hold PostScript graphics. Therefore it is easy and unproblematic to convert our PostScript file into PDF. Under Linux, use the command

```
ps2pdf -dEPSCrop <filename>.ps <filename>.pdf
```

The directive `EPSCrop` is needed to preserve the bounding box information.

Finally, lossless conversion is also possible between PostScript and the SVG vector graphic format. Only as a last resource, when an embedding application supports neither EPS nor PDF nor SVG, explicit conversion to a raster graphic may be needed. In this case, the vector graphic should be scaled to final size, or to multiple of the expected final size, before being rasterized.

2.4 PostScript

We now give a brief introduction to programming and plotting in PostScript, and thereby provides a foundation for the following section 3 about editing Frida-generated PostScript code.

PostScript was created around 1981, and got into wide-spread use thanks to consistent support by Apple and by the Unix-based open-source community. Its specification is very stable, with last additions from 1999. It is an open standard, fully documented in the “Red Book” [1] that is freely available online.

PostScript is at the same time a page description language, and a Turing-complete programming language. When introducing a programming language, it is customary to start with a trivial example program that just prints “Hello world!”. Here is one in PostScript:

```
%!PS
/Helvetica 20 selectfont
70 700 moveto
(Hello world!) show
showpage
```

When stored in a file, and opened with a PostScript viewer, it produces a page with the text “Hello world!”. The program illustrates the following properties of PostScript:

A PostScript program starts with the 4-byte identifier `%!PS`. More generally, the percent sign `%` starts a comment, which runs to the end of the line. Names, as the

font name “Helvetica”, are preceded by a backslash \. Strings are enclosed in parentheses (). These rules determine how a PostScript interpreter tokenizes a program. Valid tokens comprise names, strings, numbers, and keywords. Keywords are either built in or user defined. The hello world program contains four built-in operators, starting with `selectfont`. Non-operator tokens are successively put on top of a *stack*. Operators may take tokens from the stack, and may push tokens on the stack. For instance, the operator `selectfont` takes two arguments from the stack: a font name (here `Helvetica`), and a font size (20). The operator `moveto` takes two numbers and interprets them as x and y coordinates where to start the next drawing operation. The operator `show` takes one string from the stack, and plots it at the previously selected x, y position, in the previously selected font and size. Finally, `showpage` does not modify the stack; it transmits the current graphic to the output device.

For arithmetics, stack-based programming is known as *reverse Polish notation*. A term like $(780 - 120)/2$ can be computed in PostScript as

```
780 120 sub 2 div
```

We can assign values to variables

```
/a 4 def /b 3 def
```

and later use them:

```
a b mul
```

yields $3 \cdot 4 = 12$. To compute the hypotenuse $\sqrt{a^2 + b^2}$,

```
a dup mul b dup mul add sqrt
```

we use the operator `dup` that duplicates the topmost token on the stack. Other stack-modifying operators are

```
pop          % delete the topmost token from the stack
              %   example: A B C pop => stack becomes: A B
n copy       % duplicate the topmost n tokens
              %   A B C 2 copy => A B C B C
exch         % exchange the two topmost tokens on the stack
              %   A B C exch => A C B
n m roll     % m cyclical right shifts of the topmost n tokens
              %   A B C D E F 5 2 roll => A E F B C D
```

With these, let us write a function that converts given arguments r, φ to Cartesian coordinates:

```
/polar2xy { % r phi [requested on stack] | x y [returned on stack]
  2 copy    % r phi r phi
  cos       % r phi r cos(phi)
  mul       % r phi x [where x=r*cos(phi)]
  3 1 roll  % x r phi
  sin       % x r sin(phi)
  mul       % x y [where y=r*sin(phi)]
} def
```

The user-defined operator `polar2xy` can be used as

```
7 36 polar2xy % yields 7*cos(36) 7*sin(36)
```




Figure 1: PostScript plotting examples, drawn by the code from the end of section 2.4.

In the above function code, each line ends with a comment that shows the current stack. Such comments may help to keep an involved PostScript computation legible. Even then, legibility is a major concern in PostScript programming. Our function becomes much clearer if we store the arguments in variables instead of moving them around the stack:

```
/polar2xy { % r phi [requested on stack] | x y [returned on stack]
  /phi exch def % r
  /r exch def %
  r phi cos mul % x
  r phi sin mul % x y
} def
```

However, there is only one global variable namespace in PostScript. Therefore, a call of `polar2xy` would overwrite any previous definition of `phi` or `r`. To avoid such name clashes, temporary variables in functions must be given carefully chosen unique names. With `polar2xy_phi` and `polar2xy_r`, we would be on the safe side. However, this would make the code so clumsy that in the end we might prefer the first version of our function, with all its stack operations. Note also that functions that define variables must never be called recursively.

At this point we should explain how to debug a PostScript program if it generates incorrect graphics, or none at all. Chances are that the stack, at some point, does not contain the token sequence we thought it would. To investigate the contents of the stack, use the operator `==` in the program. It takes the topmost token from the stack, and prints a line with the name or the contents of the token to the console. Run the program `gs` (ghostscript) on the file to see the console output. To continue execution beyond the console message, use the command sequence `dup ==`.

Finally, let us demonstrate some of PostScript's plotting capabilities with the following commented code fragments. Together, they yield the image shown at reduced scale in fig. 1. A black line:

```
20 20 moveto % start path at this point
              % (coordinate origin is at bottom left)
400 0 rlineto % path goes right, in relative coordinates
0 120 rlineto % path goes up
-400 0 rlineto % path goes left
0 -90 rlineto % path goes down
stroke % draw the path in default style (thin black line)
```

Colored text, rotated by 30°:

```
1 0 0 setrgbcolor          % set color: full red, no green, no blue.
/Helvetica 24 selectfont   % set font and text size
30 30 moveto                % start drawing at this point
gsave 30 rotate             % store graphics state, and rotate by 30 deg
(Hello world!) show         % plot text
grestore                    % restore previous state (reverse rotation)
```

Connected arcs:

```
0 .5 0 setrgbcolor         % set color: dark green
10 setlinewidth            % ten times the default line width
newpath                    % reset path
210 80 45 0 270 arc        % draw arc segment around x=250, y=90 with r=60
210 80 30 270 0 arcn       % continue path with counter-clockwise arc segment
stroke
```

A filled path:

```
0 0 1 setrgbcolor          % set color: blue
340 80 2 copy               % push to stack: x y x y
moveto                      % start new path at x y
40 20 340 arc               % continue with arc segment around x y
closepath                   % close path by continuing it to the start point
fill                        % fill path in selected color
```

For more advanced PostScript programming, it is necessary to learn about conditionals (`if`, `ifelse`) and loops (`repeat`, `for`, `forall`), arrays (`[]`, `get`), graphic state (`gsave`, `grestore`), path (`newpath`, `currentpoint`, `clip`), and more. For all this, see the Red Book [1], or/and start by modifying sample code from Frida's macro collection.

3 Postprocessing Frida graphics

In section 2.2 we saw *how* to modify a PostScript file in an interactive edit session, using a text editor and an image viewer side-by-side. Let us now discuss *what* to modify in a Frida-generated PostScript graphic. We start with a structural overview before we turn to single PostScript commands that allow parameter changes or other modifications.

3.1 File structure

A Frida-generated PostScript file has three main parts:

- a dictionary with macro definitions,
- a setup section, and
- the actual plot.

The dictionary part is a verbatim copy of the file `wups11a.ps` that is installed along with Frida; depending on the installation path, it typically resides in `/usr/share/frida` or `/usr/local/share/frida`. If the graphic file is generated by the command `gf` instead of `gp`, then the dictionary part is omitted. The dictionary part mainly

contains a huge set of macro definitions. Many of these macros are used in the page setup or/and in the Frida generated plot. Many other macros are not used by Frida but provided for users who may employ them in manual enhancements of Frida generated graphics. To give just two important examples that are further explained below: There are macros for typesetting strings that contain greek letters, symbols, subscripts or superscripts (section 3.4); and there are macros for building a legend that explains the meaning of plotted symbols (section 3.5).

The setup section contains commands to control the size and aspect ratio of the plot frame, the position on the page, the background, the appearance of metadata, and the shape, size, and color of plot symbols. All this is described below in section 3.2. The entire section is a verbatim copy of the file `g3.ps` that resides in the same `share` directory as `wups11a.ps`. These file names are soft coded in the configuration file `frida.ini`. This allows users to replace `g3.ps` by a customized version of their own.

Only the third part of a Frida-generated graphic file with the actual plot is not just copied, but written line by line by Frida. It consists of three sections: an initial section that defines a plot frame with coordinate axes, ticks, numbers, and labels; a core section with the data points to be plotted; and a metadata section that explains where the data come from. All this is further elaborated in section 3.3.

For orientation in a graphic file, there are a few marks. Open a graphic file with an editor or a text viewer, and search for the string `ewu`. It will occur exactly once, in a comment block near the end of the dictionary part. And similarly, the mark `ecu` should occur no more than once, in a comment block near the end of the setup section.

3.2 Setup

The setup section in Frida graphic file is located after the mark `ewu`, and starts with a comment block that says `Customizable plot setup`, copied from `g3.ps`. Its first substantial line

```
10 dup autolabel defsiz
```

contains the operator `defsiz`, which takes two tokens from the stack to define the overall size of the figure and the base size of plot symbols and labels. The command `autolabel` performs a sublinear transformation on the topmost token on the stack. It ensures the legibility by making symbols and labels relatively *larger* for *smaller* figures. The next command

```
1 dup geld stdred
```

allows to set a magnification relative to the global scale set by `defsiz`. It performs a strictly linear scaling of the figure, leaving the relative size of symbols and labels invariant. To double the size of a figure, just replace the number 1 by 2. The interplay of `defsiz` and `defred` is illustrated in fig. 2.

Besides setting a reduction or magnification, the command `defred` also sets the aspect ratio of the plot frame. It therefore takes two arguments, that define the length of the x and y axis, respectively. The above described strictly linear scaling of the figure holds only if both arguments are kept proportional to each other, which in the default setting is accomplished by the operator `dup`. The operator `geld` multiplies

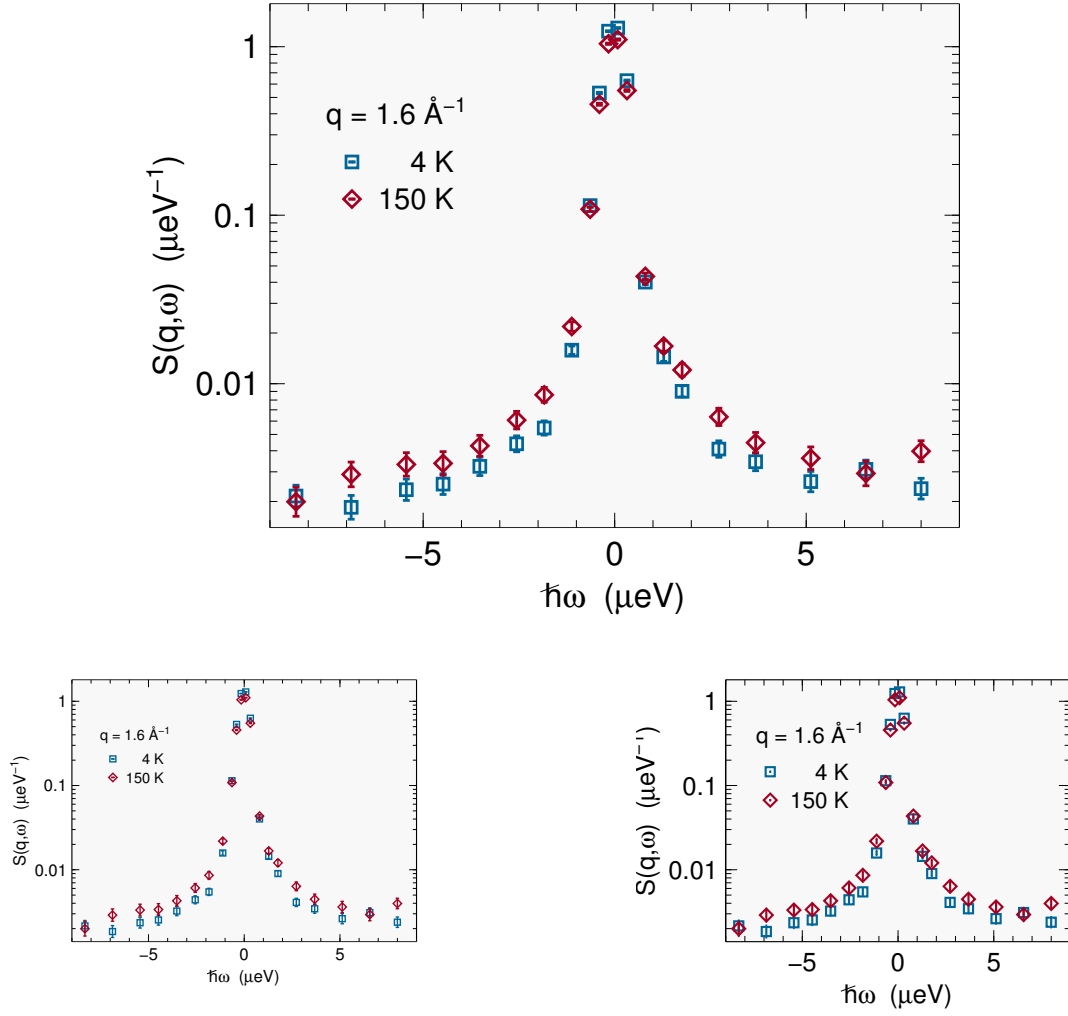


Figure 2: One and the same figure plotted with three different magnifications or/and size settings. Top: a figure with global size 12 (`defsiz`) and reduction 1 (`defred`), and scaled by the embedding text processor to 80% of the textwidth. Bottom left: exactly the same figure, but scaled to 40% of the textwidth; plot symbols, ticks, labels, and legend now look far too small. Bottom right: the same figure, except for the settings global size 6, reduction 2. Symbols, ticks, labels, legend are back to readable, thanks to the nonlinear operator `autolabel`.

<code>/gyld {0.447214 mul} def</code>	<code>/Gyld {0.447214 div} def</code>
<code>/guld {0.547723 mul} def</code>	<code>/Guld {0.547723 div} def</code>
<code>/gold {0.618034 mul} def</code>	<code>/Gold {0.618034 div} def</code>
<code>/gild {0.707107 mul} def</code>	<code>/Gild {0.707107 div} def</code>
<code>/geld {0.759836 mul} def</code>	<code>/Geld {0.759836 div} def</code>
<code>/gald {0.817765 mul} def</code>	<code>/Gald {0.817765 div} def</code>

Table 3: Predefined aspect ratio operators.

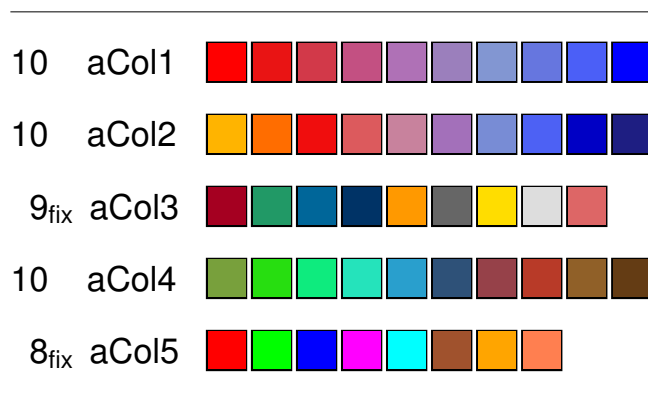


Figure 3: Predefined color sequences. Individual colors are set by the command `i ni aColX` where `i` is an integer between 0 and $ni - 1$. The number of different colors, `ni`, can be chosen arbitrarily, except if the subscript `fix` indicates the contrary.

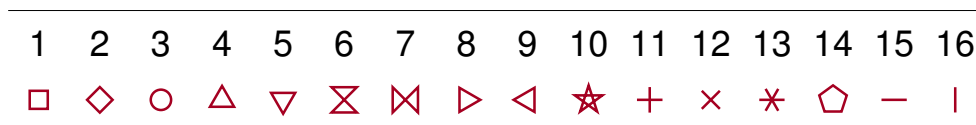


Figure 4: Predefined symbols. They are set by the command `pset`, as explained in the text.

the y argument with about 0.76, and thereby produces a moderate “landscape” frame with an aspect ratio close to 3:4. Table 3 lists other aspect ratio operators. Operators starting with a capital `G` produce “portrait” formats. The operators `gold` and `Gold` produce the overrated *golden ratio*.³ Other predefined multiplier operators are listed in table 3. To obtain a square frame, omit the operator. It is of course possible to use other multipliers. The only reason to stay with the arbitrary, but long-term stable selection of table 3 is the hope that a presentation, an article or a book will look more consistent if only a small number of different aspect ratios is used.

Next, the command

```
2 -11 setnewpage newpage
```

sets the origin (bottom left) of the plot frame relative to the A4 page. This affects only raw PostScript image, since it will be compensated as soon as a bounding box is set.

The following stance contains a few global settings. In Boolean switches like

```
1 1 InfSet
```

the digits 0 and 1 stand for no and yes. The first argument of `InfSet` determines whether to display the file name below the figure. The second argument determines whether to display all other metadata. The command

³The golden ratio naturally occurs in pentagons, in the Fibonacci series, in quasicrystals, and various other contexts. However, its importance in the arts has been hugely exaggerated [2]. There is no good reason why a rectangle with an aspect ratio of 0.618034... should be particularly pleasing, and support from empirical psychology is unconvincing at best [3]. For scientific plots, less elongated rectangles usually work better, the more so in portrait orientation.

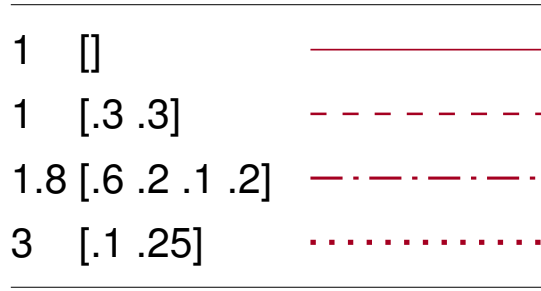


Figure 5: Dashed lines, for different arguments of `lset`.

```
1 dup 2 SymGSet
```

does global settings for the plot symbols: The first argument sets the size, the second argument sets the linewidth, which is relevant for open symbols, and the third argument determines whether to plot error bars; it has three possible values: 0 for no, 1 for yes, and 2 to let the individual `pset` commands (described below) decide. The command

```
F /pcol x def
```

sets the default to colored plot symbols; change `F` to `T` for black and white. The command

```
{ 8 aCol5 iColA } /ipCol x bind def
```

chooses a color sequence, here sequence `aCol5` with division into 8 different colors. The predefined sequences are listed in fig. 3. The command

```
/pStyles [
{ pcol {1 1 0 1. 1. pset} {11 0 0 1. 1. pset 0 ipCol} ifelse }
...
] def
```

defines an array of plot symbols. These are later accessed through the command `pstyle`. The `ifelse` clause provides different settings for black/white versus colored symbols. The command `pset` takes five arguments: a plot code, as listed in fig. 4; a switch for open (0) or filled (0) symbols; a switch whether to plot error bars (0: no, 1: yes, only respected if the third argument of `SymGSet` is 2, as described above); the symbol size (relative to the first argument of `SymGSet`); and the symbol line width (relative to the second argument of `SymGSet`). The command `ipCol` takes as argument a number between 0 and $ni - 1$, and sets a color from the previously selected sequence.

The last stanza in the setup section predefines curve styles, which can later be selected using the command `cstyle`. The switch

```
T /ccol x def
```

works as the switch `pcol` explained above, except that here the default is black and white. Change `T` to `F` for colored curves. Color preselection works as before, with `icCol` instead of `ipCol`. Finally,

```
/cStyles [
{ ccol { 1. [ ] lset } { 1. [ ] lset 0 icCol } ifelse }
...
] def
```

sets an array of curve styles. The command `lset` takes two arguments: a linewidth (relative to the global preset from `SymGSet`), and a dash pattern. The empty default pattern `[]` yields a solid line. Other patterns are shown in fig. 5.

3.3 Plot frame and data plot

The third and last part of a Frida-generated PostScript file is located after the mark "ecu". In contrast to the two previous sections, it is not copied from a source file, but written line by line by Frida's plot routine. It consists of three sections: an initial section that defines the plot frame; a core section with the data points; and a metadata section.

The plot frame section starts with the commands

```
<logflag> <lower_bound> <upper_bound> xSetCoord
<logflag> <lower_bound> <upper_bound> ySetCoord
```

These commands inform about the chosen axes. They define the operators `wx` and `wy` that can be used to transform from physical coordinates to plot coordinates. These commands should not be manually modified.

Next, the x and y axes are defined through commands like

```
/xPlotFrame {
  [
    1.875000 {(-20)}
    5.000000 {(0)}
    8.125000 {(20)}
  ] SetTacVec
  -1.25 11.25 5 4 SetTicVecLin
  {(E (ueV))} % label [often needs postprocessing]
  0 10 0 0 0 90 OneAxx Axx Tic Tac xNumL %% low x axis
  0 10 0 10 0 270 OneAxx Axx Tic Tac %% top x axis
  xCL
} def
```

The array `SetTacVec` contains a division of the axis to be used for plotting large ticks ("tacks") and number labels. Numbers are in Frida's extended string format, described in section 3.4, and can be modified accordingly. For instance, a user may wish to replace `{(0.001)}` by `{(10)(-3)sp()}` to obtain 10^{-3} . The command `SetTicVecLin` divides the axis into large and small intervals. The large division should agree with the one in `SetTacVec`. The small division then contains the tick positions (Here: divide the interval from -1.25 to 11.25 in 5 tack intervals; then divide each of them in 4 tick intervals). The command `OneAxx` takes six arguments to define an axis with international coordinates, origin, length and orientation. `Axx` actually plots the axis, `Tic` and `Tac` plot small and large ticks, and `xNumL` plots number labels below the bottom x axis. The command `xCL` plots a label below the axis. It takes one argument, here `{(E (ueV))}`, in Frida's extended string format.

The data plot section contains one block per spectrum or per curve. Each block begins with a header like

```
1 [ 249.822 ] zValues
1 pstyle
```

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
α	β	χ	δ	ε	ϕ	γ	η	ι	φ	κ	λ	μ	ν	\omicron	π	θ	ρ	σ	τ	υ	ϖ	ω	ξ	ψ	ζ
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	B	X	Δ	E	Φ	Γ	H	I	ϑ	K	Λ	M	N	O	Π	Θ	P	Σ	T	Y	ς	Ω	Ξ	Ψ	Z

Table 4: Upper row: latin characters from the “Helvetica” font. Lower row: corresponding greek characters from the “Symbol” font.

The command `zValues` informs about z coordinates. The command `pstyle` selects one of the plot styles predefined in array `pStyles`. Similarly, `cstyle` selects one of the curve styles from `cStyles`. Plotting of the following data can be disabled by setting the argument of `pstyle` or `cstyle` to 0.

3.4 Extended string format

Frida’s PostScript macros define an *extended string format* that allows users to concatenate normal text with greek characters, symbols, subscripts, and superscripts. This format is supported by the axis label commands `xCL` etc. (section 3.3) and by all other text placement commands (section 3.5).

In plain PostScript, strings are enclosed in parentheses: `(string)`. For use with Frida’s text placement operators, they must be further enclosed by curly braces: `{(string)}`. These braces may also contain additional elements according to this syntax:

`{X* (string)}`

where X^* stands for 0,1,... repeats of X . The following lines show the two different forms of X :

`(string) (string) operator`
`(string) macro`

This is best explained by an example. Suppose, we want to print “ $x = 2\pi/3$ ”. To print the greek character π , we need to switch from the default latin character font to the “Symbol” font. This accomplished by the operator `g`, and the full extended string is

`{(x=2)(p)g(/3)}`

In the “Symbol” font, the greek “ π ” occupies the same position as the letter “p” does in the latin character fonts (see table 4 for these correspondences). The operator `g` takes this character from the Symbol font, concatenates it with the preceding string “ $x = 2$ ”, and pushes “ $x = 2\pi$ ” to a special stack. Frida’s text placement command, when operating on the full extended string takes the last string “/3”, concatenates it with the contents of the special stack, and thereby generates the output “ $x = 2\pi/3$ ”.

The PostScript string syntax allows matched parentheses inside the enclosing parenthesis, like in `(distance (meters))`. However, unmatched parentheses must be

0	1	2	3	4	5	6	7	8	9	()	[]	{	}	!	?
0	1	2	3	4	5	6	7	8	9	()	[]	{	}	!	?
<hr/>																	
.	,	;	:	=	+	-	*	/	<	>	%		#	&	_	~	
.	,	;	:	=	+	-	*	/	<	>	%		#	&	_	~	
<hr/>																	

Table 5: These symbols occupy equal positions in “Helvetica” (and other fonts with standard latin encoding) and in the “Symbol” font.

\$	"	@	'	'	\
≡	∇	≡	—	∃	∴

Table 6: Keyboard symbols in “Helvetica” that correspond to completely different symbols in the “Symbol” font.

\136	\240	\241	\242	\243	\244	\245	\246	\247	\250	\251	\252	\253
⊥	€	Ÿ	'	≤	/	∞	f	♣	♦	♥	♠	↔
\254	\255	\256	\257	\260	\261	\262	\263	\264	\265	\266	\267	\270
←	↑	→	↓	°	±	"	≥	×	∞	∂	•	÷
\271	\272	\273	\274	\275	\276	\277	\301	\302	\303	\304	\305	\306
≠	≡	≈	...		—	↵	ℑ	℔	℔	⊗	⊕	∅
\307	\310	\311	\312	\313	\314	\315	\316	\317	\320	\321	\325	\326
∩	∪	⊃	⊇	⊄	⊂	⊆	∈	∉	∠	∇	Π	√
\327	\330	\331	\332	\333	\334	\335	\336	\337	\340	\341	\345	\361
·	¬	∧	∨	↔	←	↑↑	⇒	↓↓	◇	⟨	Σ	⟩

Table 7: These symbols from the “Symbol” font are accessible through octal codes.

\136	\244	\253	\273	\254	\255	\260	\261	\264	\267	\327	\367	\241	\277
^	¤	«	»	¬	-	°	±	´	·	×	÷	¡	¿
\242	\243	\245	\247	\265	\300	\301	\302	\303	\304	\305	\306	\307	\310
¢	£	¥	§	µ	À	Á	Â	Ã	Ä	Å	Æ	Ç	È
\311	\312	\313	\314	\315	\316	\317	\320	\321	\322	\323	\324	\325	\326
É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ	Ö
\330	\331	\332	\333	\334	\335	\336	\337	\340	\341	\342	\343	\344	\345
Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
\346	\347	\350	\351	\352	\353	\354	\355	\356	\357	\360	\361	\362	\363
æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ	ò	ó
\364	\365	\366	\370	\371	\372	\373	\374	\375	\376	\377			
ô	õ	ö	ø	ù	ú	û	ü	ý	þ	ÿ			

Table 8: These symbols and accentuated letters from “Helvetica” (and other fonts with standard latin encoding) are accessible through octal codes.

g	greek	{() (e)g() }	ϵ
sb	subscript	{(E)(f)sb(- E)(i)sb() }	$E_f - E_i$
sp	superscript	{(m c)(2)sp() }	$m c^2$
sbgr	greek subscript	{(K)(a)sbgr() }	K_α
spgr	greek superscript	{(n)(3-)sp()(h)spgr() }	$n^{3-\eta}$

Table 9: Frida-defined string operators, with application examples.

{()hbar() }	\hbar	{()wbar() }	$\omega/2\pi$
{()hbarw() }	$\hbar\omega$	{()taumean() }	$\bar{\tau}$
{()Sqw() }	$S(q, \omega)$	{()Angr() }	\AA^{-1}
{()ueV() }	μeV	{()inAngr() }	(\AA^{-1})
{()inueV() }	(μeV)	{()inAngrr() }	(\AA^{-2})
{()inueVr() }	$(\mu\text{eV})^{-1}$		
{()inmeVr() }	$(\text{meV})^{-1}$		

Table 10: Macros, especially for QENS applications.

escaped: `(...\(...) or (...)\(...)`. This happens in particular when greek and latin letters are mixed inside parentheses: To typeset $f(\Delta x)$, use `{(f \() (D)g(x\))}`.

Digits, interpunctuation characters, arithmetic operators, and a few more special characters from the standard latin encoding are duplicated at equal positions in the “Symbol” font. They are listed in table 5. Occasionally, this saves us from switching between normal and symbol font. For instance, we can typeset $\alpha = \pi/3 - \delta$ all in “Symbol” as `{() (a=p/3-d)g()}`. Table 6 shows how the remaining special characters from the standard keyboard translate to different special characters in the “Symbol” font. Many other characters from the “Symbol” font are accessible through octal codes, listed in table 7. For instance, `{() (\26160\260)g()}` yields “ $\pm 60^\circ$ ”. Finally, the standard latin encoded fonts also contain some useful symbols, plus many accentuated letters from European languages (table 8). For instance, use `{(\305ngstr\366)}` to typeset “Ångström”. Use the μ from the latin encoding as a unit prefix, as in `{(E (\265eV))}` for E (μeV); in this way, parentheses must not be escaped.

Besides `g` for greek letters and special symbols, the Frida PostScript header provides operators for subscripts and superscripts. All operators are listed in table 9.

Besides operators, the Frida PostScript header also provides a collection of macros that abbreviate frequently used strings. Macros insert a string, and concatenate it with the preceding and the subsequent string. Table 10 lists the most important ones. Macros can be combined as in `{()Sqw()inueVr()}` for $S(q, \omega)$ (μeV) $^{-1}$, or `{(q) (2)sp()inAngrr()}` for q^2 (\AA^{-2}).

3.5 Text placement commands

```
2 8 21 1.8 NewList
% arguments: x y fontsize linespacing
{(q = 1.2 )Angr()} TxLine
1 {(      4 K)} PtTxLine
2 {(200 K)} PtTxLine
3 {(240 K)} PtTxLine
4 {(280 K)} PtTxLine
```

For a legend, insert a list in the figure:

$q = 1.2 \text{\AA}^{-1}$

■ 4 K

◆ 200 K

▲ 240 K

▼ 280 K

Any other text can be pasted at arbitrary positions in the figure:

```
21 setown % set font size

5 5 {(centered text)} textCM
```

```

1 1 {(bottom left aligned)} textLB
9 9 {(top right aligned)} textRT

5 5 60 {(rotated by 60)(\260)g()} rtextCM

```

3.6 Concatenation, insets

Several Frida plots can be combined into one image. Please contact me if you want to use this feature. I will then either give you a private tutorial, or fill this section.

References

- [1] Adobe Systems Incorporated, *PostScript Language Reference*, Addison-Wesley: Reading, Mass. (³1999). Freely available online at <https://www.adobe.com/products/postscript/pdfs/PLRM.pdf>.
- [2] G. Markowsky, College Math. J. **23**, 2 (1992).
- [3] C. D. Green, Perception **24**, 937 (1995).

Index

- = (PostScript operator), 9
- == (PostScript operator), 9
- [(PostScript operator), 10
- % (PostScript comment), 7
-] (PostScript operator), 10

- a (Frida command), 5
- aCol1 etc. (Frida PostScript macro), 13, 14
- add (PostScript operator), 8
- Ångström
 - Frida graphic, 19
- arc (PostScript operator), 10
- arcn (PostScript operator), 10
- Aspect ratio
 - plot frame, 12, 13
- autolabel (Frida PostScript macro), 11, 12
- Axis
 - Frida graphic, 15
- Axx (Frida PostScript macro), 15

- bbox, 6
- bboxx, 6
- Bitmap, *see* Raster graphics
- Bounding box, 6

- ccol (Frida PostScript macro), 14, 15
- closepath (PostScript operator), 10
- Color
 - sequence, predefined by Frida macro, 13, 14
- Concatenation
 - Frida graphic file, 20
 - PostScript strings, 19
- copy (PostScript operator), 8
- cstyle (Frida PostScript macro), 14, 16

- Dash pattern
 - Frida graphic, 14, 15
- Debugging
 - PostScript, 6, 9
- def (PostScript operator), 8
- defred (Frida PostScript macro), 11, 12
- defsiz (Frida PostScript macro), 11, 12
- div (PostScript operator), 8
- dup (PostScript operator), 8

- ecu (mark in Frida graphic file), 11, 15
- EMF (graphic format), 4, 5
- Encapsulated PostScript (EPS), 7
- EPS (graphic format), *see* Encapsulated PostScript
- Error bar
 - Frida graphic, 14
- Evince (PostScript and PDF viewer), 6
- ewu (mark in Frida graphic file), 11
- exch (PostScript operator), 8
- EXIF (graphic format), 5
- Extended string format
 - Frida graphic file, 16

- File formats
 - graphics, 4
- fill (PostScript operator), 10
- for (PostScript operator), 10
- forall (PostScript operator), 10
- Frida
 - command
 - a, 5
 - g..., 5
 - gf, 5, 10
 - gp, 5, 10
 - p, 5
 - customization
 - frida.ini, 11
 - graphic file
 - aCol1 etc. (PostScript macro), 13, 14
 - aspect ratio, 12
 - autolabel (PostScript macro), 11, 12
 - axis, 15
 - Axx (PostScript macro), 15
 - ccol (PostScript macro), 14, 15
 - color sequence, 13, 14
 - concatenation, 20
 - cstyle (PostScript macro), 14, 16
 - defred (PostScript macro), 11, 12
 - defsiz (PostScript macro), 11, 12
 - dictionary part, 10
 - extended string format, 16
 - filename display, 13
 - g (PostScript macro), 16
 - Gald (PostScript macro), 12
 - gald (PostScript macro), 12
 - Geld (PostScript macro), 12
 - geld (PostScript macro), 12
 - Gild (PostScript macro), 12
 - gild (PostScript macro), 12

- Gold (PostScript macro), 12
- gold (PostScript macro), 12
- Guld (PostScript macro), 12
- guld (PostScript macro), 12
- Gyld (PostScript macro), 12
- gyld (PostScript macro), 12
- icCol (PostScript macro), 14
- InfSet (PostScript macro), 13
- inset, 20
- ipCol (PostScript macro), 14
- legend, 19
- lset (PostScript macro), 14, 15
- macro definitions, *see* dictionary
- part
- magnification, 11, 12
- marks, 11
- metadata display, 13
- NewList (PostScript macro), 19
- OneAxx (PostScript macro), 15
- origin, 13
- pcol (PostScript macro), 14
- plot part, 11
- plot symbol, 13, 14
- pset (PostScript macro), 13, 14
- pstyle (PostScript macro), 14, 16
- PtTxLine (PostScript macro), 19
- rtextCM (PostScript macro), 20
- sb (PostScript macro), 18
- sbgr (PostScript macro), 18
- setnewpage (PostScript macro), 13
- SetTacVec (PostScript macro), 15
- setup part, 11–15
- size, 11
- sp (PostScript macro), 18
- spgr (PostScript macro), 18
- string format, 16
- structure, 10
- SymGSet (PostScript macro), 14
- Tac (PostScript macro), 15
- text, 19
- textCM (PostScript macro), 20
- textLB (PostScript macro), 20
- textRT (PostScript macro), 20
- Tic (PostScript macro), 15
- TxLine (PostScript macro), 19
- wx (PostScript macro), 15
- wy (PostScript macro), 15
- xCL (PostScript macro), 15, 16
- xNumL (PostScript macro), 15
- xPlotFrame (PostScript macro), 15
- xSetCoord (PostScript macro), 15
- ySetCoord (PostScript macro), 15
- zValues (PostScript macro), 16
- Frida graphic
 - dash pattern, 14, 15
- frida.ini, 11
- g (Frida PostScript macro), 16
- g3.ps, 11
- g...(Frida command), 5
- Gald (Frida PostScript macro), 12
- gald (Frida PostScript macro), 12
- Geld (Frida PostScript macro), 12
- geld (Frida PostScript macro), 12
- get (PostScript operator), 10
- gf (Frida command), 5, 10
- Ghostscript (PostScript viewer), 6
- GIF (graphic format), 5
- Gild (Frida PostScript macro), 12
- gild (Frida PostScript macro), 12
- Gnuplot, 5
 - Frida graphic backend, 4
- Gold (Frida PostScript macro), 12
- gold (Frida PostScript macro), 12
- Golden ratio, 13
- gp (Frida command), 5, 10
- Graphics
 - file formats, 4
 - postprocessing
 - workflow, 5
 - raster, 4
 - vector, 4
- Greek character
 - Frida graphic, 18
 - Frida graphic file, 16
- grestore (PostScript operator), 10
- gsave (PostScript operator), 10
- Guld (Frida PostScript macro), 12
- guld (Frida PostScript macro), 12
- Gyld (Frida PostScript macro), 12
- gyld (Frida PostScript macro), 12
- Helvetica
 - PostScript font, 8, 10, 16–18
- icCol (Frida PostScript macro), 14
- if (PostScript operator), 10
- ifelse (PostScript operator), 10
- InfSet (Frida PostScript macro), 13
- Inset
 - Frida graphic file, 20
- ipCol (Frida PostScript macro), 14

- JPG (graphic format), 5
- LaTeX
 - supported graphic formats, 6
- Legend
 - Frida graphic, 19
- Libreoffice
 - supported graphic formats, 6
- lset (Frida PostScript macro), 14, 15
- Macro
 - Frida graphic, 18
- Magnification
 - Frida graphic file, 11, 12
- Metadata
 - display in Frida graphic file, 13
- Microsoft Office
 - supported graphic formats, 6
- moveto (PostScript operator), 8, 10
- mul (PostScript operator), 8
- Neutron scattering
 - String macros, in Frida graphic, 18
- NewList (Frida PostScript macro), 19
- newpath (PostScript operator), 10
- Office (Microsoft)
 - supported graphic formats, 6
- OneAxx (Frida PostScript macro), 15
- Origin
 - Frida graphic file, 13
- p (Frida command), 5
- PBM (graphic format), 5
- pcol (Frida PostScript macro), 14
- PDF (graphic format), *see* Portable Document Format
- PGM (graphic format), 5
- Pixel map, *see* Raster graphics
- Plot
 - PostScript file, *see* Frida, graphic file
- Plot symbol
 - Frida graphic, 13, 14
- PNM (graphic format), 5
- pop (PostScript operator), 8
- Portable Document Format (PDF), 7
 - conversion from PostScript, 7
 - viewer
 - Evince, 6
- PostScript
 - arrays, 10
 - bounding box, 6
 - comments, 7
 - conversion to EPS, 7
 - conversion to PDF, 6
 - debugging, 6, 9
 - editing, 6
 - font
 - Helvetica, 8, 10, 16–18
 - latin encoding, 16–18
 - Symbol, 16, 17, 19
 - fonts, 8
 - Frida-generated, *see* Frida, graphic file
 - graphic state, 10
 - names, 8
 - operator
 - =, 9
 - ==, 9
 - [, 10
 -], 10
 - add, 8
 - arc, 10
 - arcn, 10
 - closepath, 10
 - copy, 8
 - def, 8
 - div, 8
 - dup, 8
 - exch, 8
 - fill, 10
 - for, 10
 - forall, 10
 - get, 10
 - grestore, 10
 - gsave, 10
 - if, 10
 - ifelse, 10
 - moveto, 8, 10
 - mul, 8
 - newpath, 10
 - pop, 8
 - repeat, 10
 - rlineto, 9
 - roll, 8
 - selectfont, 8
 - setlinewidth, 10
 - setrgbcolor, 10
 - show, 8
 - showpage, 8
 - sqrt, 8
 - stroke, 9
 - sub, 8
 - path, 10

- programming
 - hello world, 7
- rotation, 10
- stack, 8
- standard, 7
- viewer
 - Evince, 6
 - written by Frida, 4, 5
- Powerpoint (Microsoft), *see* Office (Microsoft)
- PPM (graphic format), 5
- PS (graphic format), *see* PostScript
- ps2pdf, 7
- pset (Frida PostScript macro), 13, 14
- pstyle (Frida PostScript macro), 14, 16
- PtTxLine (Frida PostScript macro), 19
- QENS
 - String macros, in Frida graphic, 18
- Raster graphics, 4
 - conversion from vector format, 7
- Rectangle
 - plot frame, aspect ratio, 13
- repeat (PostScript operator), 10
- rlneto (PostScript operator), 9
- roll (PostScript operator), 8
- rtextCM (Frida PostScript macro), 20
- sb (Frida PostScript macro), 18
- sbgr (Frida PostScript macro), 18
- selectfont (PostScript operator), 8
- setlinewidth (PostScript operator), 10
- setnewpage (Frida PostScript macro), 13
- setrgbcolor (PostScript operator), 10
- SetTacVec (Frida PostScript macro), 15
- show (PostScript operator), 8
- showpage (PostScript operator), 8
- Size
 - Frida graphic file, 11
- sp (Frida PostScript macro), 18
- spgr (Frida PostScript macro), 18
- sqrt (PostScript operator), 8
- String
 - Frida graphic
 - macros, 18
- String format
 - Frida graphic file, 16
- stroke (PostScript operator), 9
- sub (PostScript operator), 8
- Subscript
 - Frida graphic, 18
- Superscript
 - Frida graphic, 18
- SVG (graphic format), 5, 7
- Symbol
 - Frida graphic, 13, 14
 - PostScript font, 16, 17, 19
- SymGSet (Frida PostScript macro), 14
- Tac (Frida PostScript macro), 15
- Text
 - Frida graphic, 19
- textCM (Frida PostScript macro), 20
- textLB (Frida PostScript macro), 20
- textRT (Frida PostScript macro), 20
- Tic (Frida PostScript macro), 15
- TIFF (graphic format), 5
- TxLine (Frida PostScript macro), 19
- Vector graphics, 4
- Viewer
 - Evince, 6
- WMF (graphic format), 4, 5
- Word (Microsoft), *see* Office (Microsoft)
- wups11a.ps, 10
- wx (Frida PostScript macro), 15
- wy (Frida PostScript macro), 15
- xCL (Frida PostScript macro), 15, 16
- xNumL (Frida PostScript macro), 15
- xPlotFrame (Frida PostScript macro), 15
- xSetCoord (Frida PostScript macro), 15
- yCL
 - see* xCL (Frida PostScript macro), 15
- yPlotFrame
 - see* xPlotFrame (Frida PostScript macro), 15
- ySetCoord (Frida PostScript macro), 15
- zValues (Frida PostScript macro), 16