

Fig → `<svg>`
`1 0 obj`
`prologues:=0;`
`%% BoundingBox`

The fig2vect manual

Dipl.-Ing. D. Krause

November 18, 2009

Contents

1	User manual	5
1.1	Overview	5
1.1.1	Purpose	5
1.1.2	Why fig2vect?	6
1.1.3	License	8
1.2	Installation	9
1.2.1	Installation on Unix/Linux systems	9
1.2.2	Installation on Windows systems	10
1.2.3	Installation SVG variants of the Ghostscript fonts	11
1.3	Program usage	13
1.3.1	Drivers	13
1.3.2	Program invokation	26
1.3.3	Options	27
1.4	Configuration mechanisms	29
1.4.1	Configuration file	29
1.4.2	Command line arguments	30
1.4.3	Control comments (special comments)	30
1.5	Configuration entries	32
1.5.1	For all drivers	32
1.5.2	MetaPost	41
1.5.3	EPS/PS	43
1.5.4	PDF	46
1.5.5	TeX	47
1.5.6	SVG	48
1.6	Additional hints	51
1.6.1	Background rectangles	51
1.6.2	How to use fill patterns	52

1.6.3	Embedded images	53
1.6.4	SVG images	54
1.7	Bitmap images	55
1.7.1	Overview	55
1.7.2	Detailed conversion steps	56
2	Technical details	59
2.1	Mathematical aspects of the fig2vect program	59
2.1.1	Before you read this section	59
2.1.2	Curves, X-splines and Bezier-splines	59
2.1.3	Arcs	69
2.1.4	Arrowheads	71
2.2	Notes	77
2.2.1	Font handling	77

Chapter 1

User manual

1.1 Overview

1.1.1 Purpose of the program

The `fig2vect` program converts `*.fig` files – created using XFig, jFig or WinFig – into other vector graphic files (METAPOST, PS/EPS, PDF or SVG), see figure 1.1. METAPOST, PS/EPS and PDF can be used in \LaTeX / pdf\LaTeX documents, SVG files can be used for web publishing.

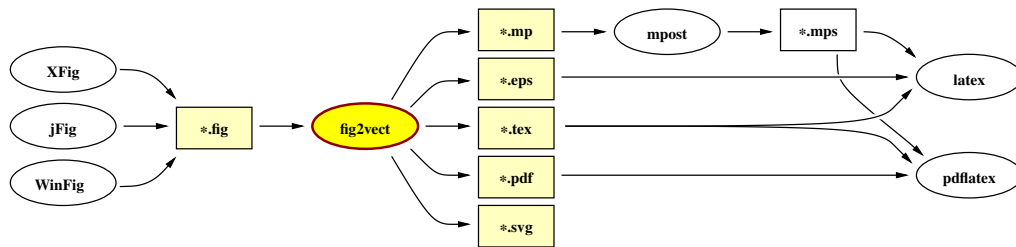


Figure 1.1: Purpose of the program

1.1.2 Differences to transfig

Note: All statements about transfig or XFig in this document are related to version 3.2.4.

Both fig2vect and transfig can be used to convert *.fig files into other file formats. There is a number of differences between the programs:

- **Different audience.**
 Transfig is a general conversion program from Fig to other file formats and provides a large number of possible output formats. A wide range of image file formats for embedded graphics can be used.
 Fig2vect is specialized for use with L^AT_EX/pdfL^AT_EX and provides only output formats used in such environments. Only PNG, JPEG and NETPBM files can be used for embedded images, the EPS/PS driver can also use EPS/PS images if they contain a “%%BoundingBox” information.
- **Internal program structure.**
 Both transfig and fig2vect can be modified to add new drivers. A transfig driver consists of a set of functions, each function is invoked to handle one drawing primitive (dt.: Graphikelement).
 A fig2vect driver is a function which is invoked on the entire drawing. The function can traverse the collection of drawing primitives as often as necessary, i.e. to collect information in the first pass and to do the real output in a second pass.
- **Fill patterns are vector graphics.**
 In fig2vect fill patterns are drawn using vector graphics operations instead of bitmap images.
- **Spline handling.** Transfig converts X-splines into polylines while reading the Fig file. Fig2vect uses a number of bezier spline segments to approximate an X-spline segment.
- **Image width and height.**
 Some transfig drivers use the paper size from the Fig file header to estimate the image width and height.
 Fig2dev inspects the drawing primitives (except texts) to find the image area.

- Arrowheads.
If arrowheads are attached to lines, `fig2vect` shorts the lines to have the arrowhead ending in the specified endpoint of the line.
Arrowhead line thickness is ignored, arrowheads are drawn in the same thickness as the line itself.
- Configuration.
`Transfig` is controlled using command line parameters. `Fig2vect` uses a configuration file, only one command line argument is necessary to choose a configuration.
- No automatic rotation of included images.
If the first polygon point for embedded images is not the upper left point, `XFig` and `jFig` behave differently: `XFig` rotates the embedded image by multiples of 90° , `jFig` does not rotate the image. `Fig2vect` behaves like `jFig` and does not rotate the image.
If you really want to rotate an embedded image I suggest you do the rotation in an external graphics program and embed the image specifying the upper left point of the image area first. This creates a portable Fig file which can be edited in both `XFig` and `jFig` and can be processed by both `transfig` and `fig2vect`.

1.1.3 License conditions

The software is licensed to you under BSD style license conditions:

- Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
 - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
 - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
 - Neither the name of the Dirk Krause nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
- This software is provided by the copyright holders and contributors “as is” and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed.

In no event shall the copyright owner or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

1.2 Installation

1.2.1 Installation on Unix/Linux systems

Prerequisites

A number of libraries is required or recommended to install fig2vect:

- zlib
- libzip2
- dklibs
- libpng
- jpeglib
- netpbm

The fig2vect FAQ¹ lists all the libraries and sources to download them. Most modern UNIX/Linux distributions should allow to install the libraries (except dklibs) via package management.

Note: You need both the libraries as binary files and the header files to use the libraries with your C compiler. In some Linux distributions the header files are in separated packages, named i.e. “zlib developer support”...

Make sure compiler and linker are set up properly to find both the header files and the libraries.

Download

The primary source for fig2vect documentation and source code is the project homepage².

¹<http://fig2vect.sourceforge.net/faq.html#h5uwhich>

²<http://fig2vect.sourceforge.net>

Installation procedure on Unix/Linux systems

After unpacking the software run

```
1 ./configure
2 make
3 make install
```

as usual.

RPM preparation

To prepare RPM creation – or the creation of a packet for other software management systems – you have to install all the files into a special package source directory instead of the usual directories.

Use

```
1 ./configure
2 make
3 make pp=/tmp/packages/fig2vect install
```

to do so. The “pp” means “package prefix” here.

1.2.2 Installation on Windows systems

The dklibs SourceForge project³ provides executable setup files to install dklibs and some programs using these libraries (fig2vect is one of them).

The “dklibs-win32-*-user.exe” file installs software and documentation, “dklibs-win32-*-dev.exe” additionally install the sources for the dklibs libraries, the programs and all other libraries used to build the programs. The “dklibs-win32-*-dev.exe” file is significantly larger than the “dklibs-win32-*-user.exe” file.

³<http://sourceforge.net/projects/dklibs>

1.2.3 Installation SVG variants of the Ghostscript fonts

This step is only necessary if you want to create SVG files and you want to use exactly the fonts used in the Fig file.

The SVG standard requires SVG viewer applications to render text with attributes “serifes”/“no serifes”/“monospaces”, “normal weight”/“bold” and “up-right”/“italic”. By default fig2vect translates the exact font specifications from the Fig file into the appropriate SVG text attributes.

If you want to use exactly the same fonts as specified in the Fig file you must provide the fonts as SVG font files, the SVG graphics file contains references to the SVG font files. Before you write references to the SVG font files you have to create them. You need:

- the FontForge program⁴,
- the Ghostscript fonts,
- the Ghostscript fonts sources⁵,
- the files “comment.txt”, “crfonts.csh” and “sfd2svg.pe” from the “scripts” directory in the fig2vect source archive.

The procedure is:

- Install FontForge.
- Create a temporary directory, i.e. “/tmp/xxx”.
- Unpack the Ghostscript fonts and the Ghostscript font sources into the temporary directory. The /tmp/xxx directory should now contain a lot of *.pfb and *.sfd files.
- Copy the three files from the “scripts” directory into the temporary directory.
- Run the script

```
1 ./crfonts.csh
```

⁴<http://sourceforge.net/projects/fontforge>

⁵<ftp://ftp.gnome.ru/fonts/sources>, file “urw-fonts-1.0.7pre40-src.tar.bz2” or newer

to create the SVG font files.

- Edit all the *.svg files manually in a text editor. Insert the contents of the “comment.txt” file between the “<metadata>...</metadata>”- and the “<defs>...</defs>”- section.
In the “<metadata>...</metadata>” below the “Created by FontForge...” remove the line containing “By...” and your name/login name.
- Create a target directory to install the SVG fonts. Copy the following files into the target directory:
 - all the *.svg files,
 - the archives containing the Ghostscript fonts and the Ghostscript fonts sources and
 - the three files from the “scripts” directory.

Note: The Ghostscript fonts were published by URW under the terms of the GNU General Public License (GPL). One of the license conditions grants each user of GPL-licensed software the right to obtain and modify the source code. If you make GPL-licensed software available to other users, you are responsible to make the sources available too.

If you provide the SVG variants of the fonts – i.e. on a multi-user computer system or via a web site – you have to provide all the sources needed to build these fonts. Access to the sources must not be more complicated than access to the fonts itself. To fulfill these demands you should place the sources in the same directory as the fonts.

All statements about the GNU General Public License in this note are just an interpretation based on my knowledge about this license. They might contain errors. The full license conditions can be found in the COPYING file in the Ghostscript Font distribution or on the FSF (Free Software Foundation) web site.

1.3 Program usage

1.3.1 Drivers

Choosing a driver

Which driver to choose depends on

- the purpose of the drawing and
- the contents of the Fig file.

To prepare a graphics for a web site, use the SVG format.

To use the graphic in a \LaTeX / pdf\LaTeX document, use the MetaPost driver. MetaPost output can be used with both \LaTeX and pdf\LaTeX .

The MetaPost driver can not be used if the Fig file contains embedded images, use either the EPS/PS driver (when using \LaTeX) or the PDF driver (when using pdf\LaTeX) instead.

The EPS/PS driver can only handle non-special text, the PDF driver can only handle non-special left-aligned text in the 14 PDF base fonts. If the Fig file requires better text handling capabilities, use the combination of EPS- and \TeX -driver or PDF- and \TeX -driver.

MetaPost driver

Using the `fig2vect` MetaPost driver you create a file `*.mp`. This file is processed by the `mpost` program (on some systems named `mp`) to create a `*.0` file. This file must be renamed to `*.mps`.

The `*.mps` file can be used both by \LaTeX /`dvips` and pdf\LaTeX . A MetaPost file can contain both “normal” and special text. Special text typesetting is done by \LaTeX , for non-special text you can choose to either use \LaTeX for typesetting or simple PostScript labels.

On a Unix/Linux system use the commands below to create a file `example.mps` from `example.fig`:

```
1 fig2vect -lmp example.fig example.mp
2 mpost --tex=latex example
3 mv example.0 example.mps
```

On a Windows command line prompt use:

```
1 fig2vect -lmp example.fig example.mp
2 mpost --tex=latex example
3 xcopy example.0 example.mps
4 del example.0
```

In the \LaTeX source write the following code to include the image:

```
1 \begin{figure}
2   {\centering
3    \includegraphics{example.mps}
4    \caption{This is an image}
5    \label{fig:example}
6   }
7 \end{figure}
```

Note: The command to run MetaPost is “mpost --tex=latex ...” on some systems, “mpost -tex=latex ...” on other systems.

Use “mpost --help” to view the MetaPost help text showing the correct usage for your system.

EPS/PS driver

Use the command

```
1 fig2vect -leps example.fig example.eps
```

or

```
1 fig2vect -lps example.fig example.ps
```

to convert `example.fig` to `example.eps` or `example.ps`. In the \LaTeX source write

```
1 \begin{figure}
2 {\centering
3 \includegraphics{example}
4 \caption{This is an image}
5 \label{fig:example}
6 }
7 \end{figure}
```

to embed the image.

The EPS driver can process embedded PNG, JPEG, NETPBM and PSEPS images.

If the Fig file contains special text you must use the combination of EPS- and \TeX -driver, see section 1.3.1 on page 17.

PDF driver

Use

```
1 fig2vect -lpdf example.fig example.pdf
```

to convert `example.fig` into `example.pdf`. In the \LaTeX source (which you process using `pdf \LaTeX`) write

```
1 \begin{figure}
2 {\centering
3 \includegraphics{example}
4 \caption{This is an image}
5 \label{fig:example}
6 }
7 \end{figure}
```

to embed the image. You may have noticed that this code is the same as to embed an EPS image for use with \LaTeX /dvips.

If the Fig file embeds a PNG file with alpha channel, the alpha channel is converted into PDF transparency data.

The PDF driver can only handle non-special left-aligned text in the 14 PDF base fonts. If the Fig file contains any other text, the combination of PDF- and T_EX- driver must be used, see section 1.3.1 on page 20.

Combined EPS- and TeX drivers

In this combination the TeX driver handles special text and optionally non-special text too. The EPS driver handles all other graphic elements.

Use the commands

```
1 fig2vect -leps.tex example.fig example.eps
2 fig2vect -ltex example.fig example.tex
```

to create two files `example.tex` and `example.eps`. In the L^AT_EX source use

```
1 \begin{figure}
2 {\centering
3 \input{example.tex}
4 \caption{This is an image}
5 \label{fig:example}
6 }
7 \end{figure}
```

to embed the image. Optionally you can use a “`resizebox`” to scale the image:

```
1 \begin{figure}
2 {\centering
3 \resizebox{\linewidth}{!}{\input{example.tex}}
4 \caption{This is an image}
5 \label{fig:example}
6 }
7 \end{figure}
```

The first argument chooses the image width, the second arguments sets the image height. The exclamation mark in the example used for the image height sets up the same scale factor as used for the image width.

Figure 1.2 on page 19 shows how the EPS- and the TeX-file come together:

- The L^AT_EX source includes the `example.tex` file.
- In `example.tex` there is a first picture environment, no area is reserved on the page for this picture. The `example.eps` image is inserted on the page at the origin of the picture environment.
- A second picture environment is created, width and height are exactly the same as image width and height. So this picture environment is placed exactly over the `example.eps` image.
The labels for special texts and optionally for non-special texts are set in this picture environment.

If you want the \TeX driver to set both special and non-special text the entries in the configuration file (see section 1.4.1 on page 29) must look like this:

```

1 [ eps . tex ]
2 normal text      =      handling : tex , font : similar , size : fig , mbox
3 special text     =      font : similar , size : fig , mbox
4 skip all texts   =      yes
5 [ tex ]
6 normal text      =      handling : tex , font : similar , size : fig , mbox
7 special text     =      font : similar , size : fig , mbox

```

The “normal text = handling:tex...” in the “[eps.tex]” section chooses \LaTeX to handle non-special text. As the EPS driver doesn’t know how to do that, it ignores these texts.

The “normal text = handling:tex...” in the “[tex]” section tells the \TeX driver it has to process non-special text.

If you want the EPS driver to handle non-special text the sections must look like this:

```

1 [ eps . tex ]
2 normal text      =      handling : none , font : fig , size : fig
3 special text     =      font : similar , size : fig , mbox
4 [ tex ]
5 normal text      =      handling : none
6 special text     =      font : similar , size : fig , mbox

```

The “normal text = handling:none...” in the “[eps.tex]” section means “no external handling necessary, the EPS driver can handle the text itself”.

The “normal text = handling:none...” in the “[tex]” section sets up the \TeX driver to ignore non-special texts.

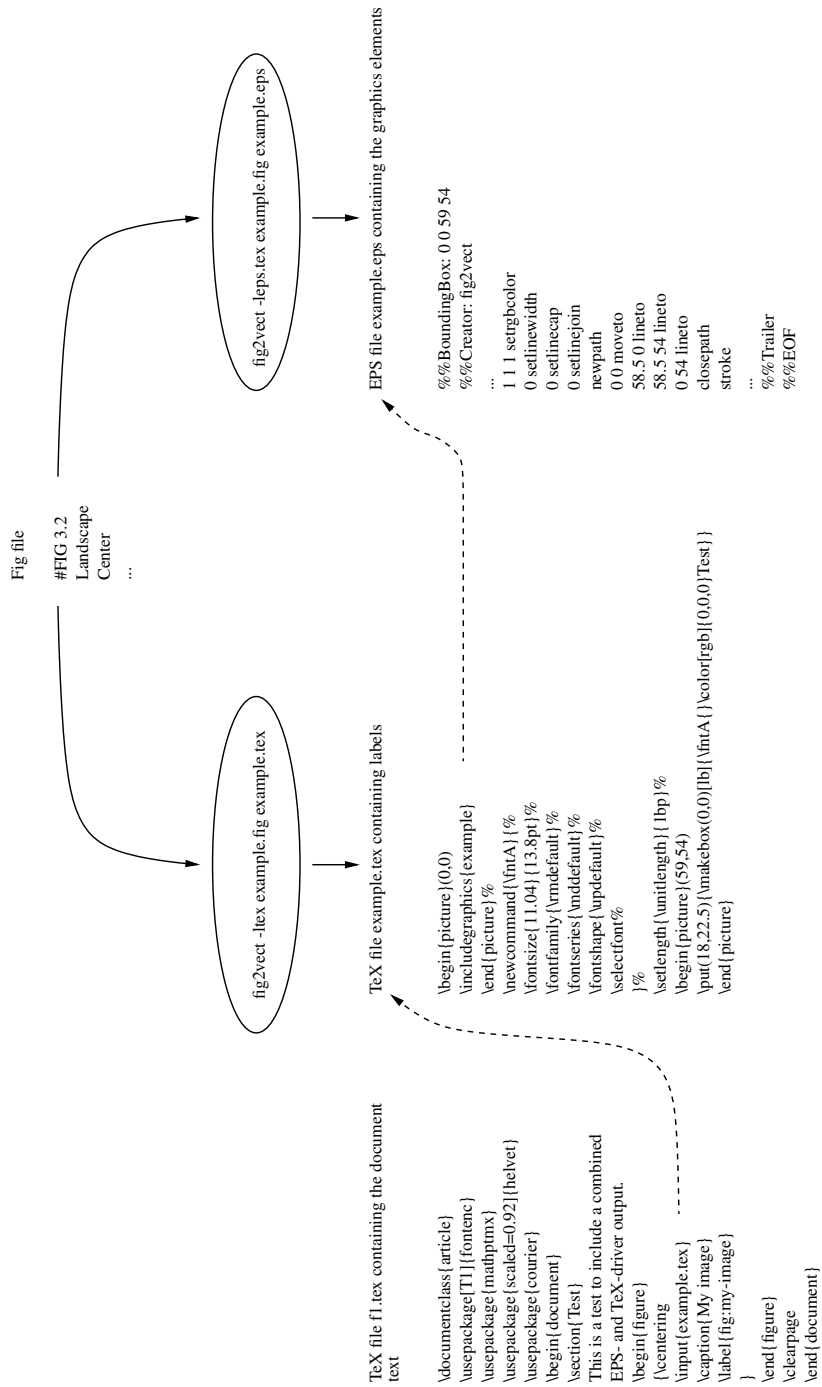


Figure 1.2: Combination of EPS and TeX driver

Combined PDF- and TeX drivers

In this combination the TeX driver handles special text and optionally non-special text too. The PDF driver handles all other graphic elements.

Use the commands

```
1 fig2vect -lpdf.tex example.fig example.pdf
2 fig2vect -ltex example.fig example.tex
```

to create two files `example.tex` and `example.pdf`. In the L^AT_EX source use

```
1 \begin{figure}
2 {\centering
3 \input{example.tex}
4 \caption{This is an image}
5 \label{fig:example}
6 }
7 \end{figure}
```

to embed the image. Optionally you can use a “`\resizebox`” to scale the image:

```
1 \begin{figure}
2 {\centering
3 \resizebox{\linewidth}{!}{\input{example.tex}}
4 \caption{This is an image}
5 \label{fig:example}
6 }
7 \end{figure}
```

Figure 1.3 on page 22 shows how the PDF- and the TeX-file come together:

- The L^AT_EX source includes the `example.tex` file.
- In `example.tex` there is a first picture environment, no area is reserved on the page for this picture. The `example.pdf` image is inserted on the page at the origin of the picture environment.
- A second picture environment is created, width and height are exactly the same as image width and height. So this picture environment is placed exactly over the `example.pdf` image.

The labels for special texts and optionally for non-special texts are set in this picture environment.

If you want the TeX driver to set both special and non-special text the entries in the configuration file (see section 1.4.1 on page 29) must look like this:

```

1 [pdf.tex]
2 normal text      =      handling:tex , font:similar , size:fig ,mbox
3 special text    =      font:similar , size:fig ,mbox
4 skip all texts  =      yes
5 [tex]
6 normal text     =      handling:tex , font:similar , size:fig ,mbox
7 special text    =      font:similar , size:fig ,mbox

```

The “normal text = handling:tex...” in the “[pdf.tex]” section chooses \LaTeX to handle non-special text. As the PDF driver doesn’t know how to do that, it ignores these texts.

The “normal text = handling:tex...” in the “[tex]” section tells the \TeX driver it has to process non-special text.

If you want the PDF driver to handle non-special text the sections must look like this:

```

1 [pdf.tex]
2 normal text     =      handling:none , font:fig , size:fig
3 special text    =      font:similar , size:fig ,mbox
4 [tex]
5 normal text     =      handling:none
6 special text    =      font:similar , size:fig ,mbox

```

The “normal text = handling:none...” in the “[pdf.tex]” section means “no external handling necessary, the PDF driver can handle the text itself”.

The “normal text = handling:none...” in the “[tex]” section sets up the \TeX driver to ignore non-special texts.

Note: The typesetting capabilities of the PDF driver are minimal: It can only typeset non-special left-aligned text in the 14 PDF base fonts. It is strongly recommended to use the \TeX driver for setting non-special text.

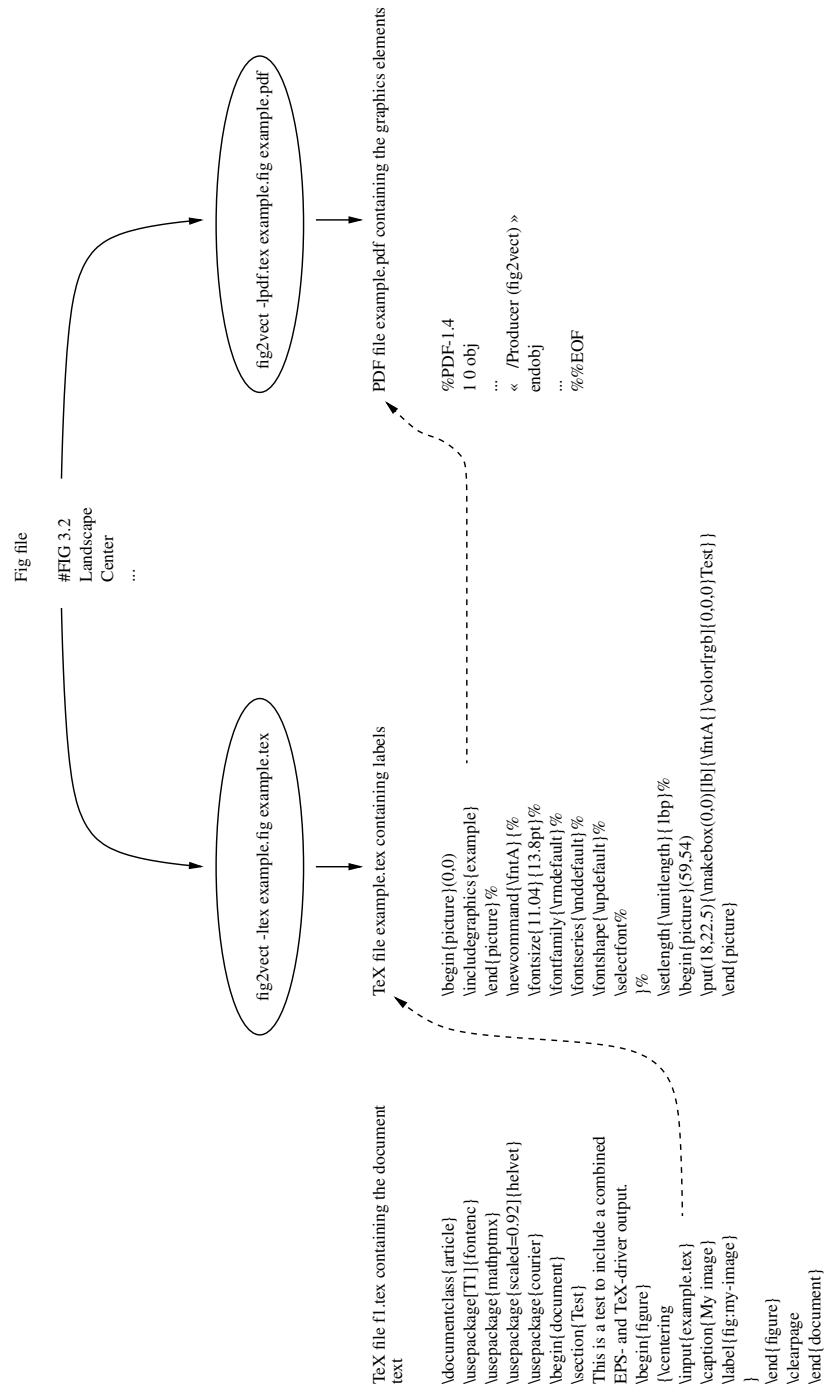


Figure 1.3: Combination of PDF- and TeX-driver

Building standalone PDF files using PDF- and TeX driver

In the previous section we used the combination of PDF and TeX driver to convert a Fig image for inclusion in L^AT_EX documents.

In this section we will create a standalone PDF image from the Fig file.

The following files are involved:

- the PDF version of the Fig image without texts, created by the fig2vect PDF driver,
- The TeX source for pdfL^AT_EX created by the fig2vect TeX driver and
- the final PDF file created by pdfL^AT_EX.

We have two PDF files here, it is important to use different names for these files. We can either

- add an “s” (for standalone) to the final PDF files base name

```
1 fig2vect -l pdf.tex example.fig example.pdf
2 fig2vect -l tex.full -o preamble.file =... example.fig examples.tex
3 pdflatex examples.tex
```

or

- add an “i” (for included) to the PDF driver output files base name.

```
1 fig2vect -l pdf.tex example.fig examplei.pdf
2 fig2vect -l tex.full -i examplei -o preamble.file =... \
3 example.fig example.tex
4 pdflatex example
```

Combination of MetaPost and T_EX driver

In this combination the T_EX driver typesets all texts, the MetaPost driver creates all other graphics elements. Use

```

1 fig2vect -lmp.tex example.fig example.mp
2 mpost --tex=latex example
3 mv example.0 example.mps
4 fig2vect -ltex -i example.mps example.fig example.tex

```

to create example.mps and example.fig.

In the L^AT_EX source use

```

1 \begin{figure}
2 {\centering
3 \input{example.tex}
4 \caption{Test image}
5 \label{fig:example}
6 }
7 \end{figure}

```

to import the image. The “resizebox” instruction can be used to resize the image if necessary.

MetaPost can handle both normal and special text, but creating the *.mpx file, running L^AT_EX and processing a *.dvi file for each image takes time. The combination of MetaPost and T_EX driver can reduce the amount of time needed – especially if a L^AT_EX source references a large number of images.

SVG driver

Use

```
1 fig2vect -lsvg example.fig example.svg
```

to create a file `example.svg` from `example.fig`. This file can be embedded into a HTML file using

```
1 <noembed>
```

```
2 
```

```
3 </noembed>
```

```
4 <embed src="example.svg" type="image/svg+xml" width="840" height="220">
```

Of course you need to set the correct width and height values.

The SVG driver can not handle special text, image flipping is ignored.

1.3.2 Program invocation

Use

```
fig2vect options directory  
fig2vect options inputfile outputfile
```

to start the program.

If you use the MetaPost driver you can also use the

```
fig2vect options  
fig2vect options inputfile
```

commands.

If there is no *inputfile* specified the program runs as filter and processes standard input.

If the program is run on a directory, the directory is traversed. For each *.fig file found the program does a conversion and creates the appropriate *.mp-, *.eps-, *.pdf-, *.tex-, *.svg- or *.tex-file (depending on the driver choosed). If “make-style” mode is activated the program compares the modification timestamps and runs a conversion only if it is necessary.

1.3.3 Options

Program options

- *-l configuration*
chooses a driver or a configuration specified in the configuration file.
- *-o key=value*
overwrite a configuration entry.
- *-m*
activates “make-style” mode.
- *-m-*
deactivates “make-style” mode.
- *-a*
chooses an output file name automatically if no output file name was specified. The file name is based on the input file name and the output driver.

Help and version

- *-h*
--help
shows a help text.
- *-v*
--version
shows the version number.

Permanent options handling

- `-c ...`
`--configure ...`
saves permanent options.
- `-u`
`--unconfigure`
removes all user-specific permanent options.
- `-C`
`--show-configuration`
shows the permanent options.
- `-r`
`--reset`
deactivates the user-specific permanent options for one program invocation.
If this option is used, it must be written before any other option.

1.4 Configuration mechanisms

1.4.1 Configuration file

File name

By default the `fig2vect` program uses the configuration file “`/${prefix}/etc/fig2vect/fig2vect.cfg`” or “`%WINDIR%\fig2vect\fig2vect.cfg`”.

A user-specific configuration file can be created as “`$/HOME/.defaults/fig2vect.cfg`” or “`%USERPROFILE%\defaults\fig2vect.cfg`”, this file is used *instead* of the system-specific configuration file.

To find out where the program searches for the configuration file and which configuration file is used, add the option “`--/log/stderr/level=debug`” to the command line arguments.

Configuration file structure

The configuration file consists of sections, each section describes one configuration. A section is introduced by the configuration name in square brackets, the remainder of the section is a list of configuration entries, one per line. Each entry consists of the entry name (multipart string) and the value, separated by “`=`”.

The first part of each configuration name is the name of the driver to use. After the driver name a unique identifier follows, driver name and identifier are separated by a dot.

An inheritance mechanism is used to grant a better overview. If you specify a command line, i.e.

```
1  fig2vect -l mp.pdf test.fig test.mp
```

the program processes the general configuration section “[*]” first. In the next step the program processes the section belonging to the driver name (“[mp]”). The section for the name specified on the command line (“[mp.pdf]”) is processed in the last step. In each step the settings from former steps may be overwritten.

Only the first dot separates the driver name from the additional identifier, adding further dots will not result in additional inheritance levels.

1.4.2 Command line arguments

Configuration entries can be overwritten by command line arguments, the command line arguments are processed after the configuration file.

To overwrite the configuration file entry

```
1 use metapost arrowheads = yes
```

one can use the command line argument

```
1 -o use . metapost . arrowheads=no
```

The parts of the entry name are separated by dots instead of spaces here.

1.4.3 Control comments (special comments)

Control comments in the Fig file are another mechanism to overwrite configuration entries. Control comments in the global comment range of the Fig file are valid for the entire file. Control comments for a graphic element only have effect on the element.

A control comment consists of

- two comment markers “#”, optional spaces are allowed,
- a list of drivers for which the control comment is valid,
- a colon and
- the configuration entry.

The example entry

```
1 ## pdf: tiled patterns = no
```

tells the PDF driver to use one large pattern cell but is ignored by other drivers.

The entry

```
1 ## *: font scale factor = 0.95
```

establishes a font scaling factor for all drivers.

If a comment is used by multiple drivers, the driver names are separated by comma, i.e.

```
1 ## mp, pdf: fill pattern = none
```

To enter a control comment, use the edit function for an object and type into the “Comments” field:

```
1 # pdf: tiled patterns = no
```

Note: You have to write only one comment marker, the second one is added by XFig automatically when saving the file.

Control comments at the document level must be added manually, place them between line 8 (transparent color) and 9 (resolution and origin).

You may use `doc2.fig`⁶ as an example, control comments to use ECMA script (JavaScript) in SVG were added to this file.

⁶<http://fig2vect.sourceforge.net/doc2.fig>

1.5 Configuration entries

1.5.1 Configuration entries for all drivers

General configuration entries

- **lighten look** = *boolean:lighten*
chooses, whether or not to lighten the look by using only a half of the line width specified in the Fig file for all lines.
- **web palette** = *boolean:web-palette*
chooses, whether or not (default) to use a web-optimized color palette instead of normal colors. This option is turned off by default.
- **use cs setting** = *boolean:cs-setting*
chooses, whether or not to obtain the origin setting from the Fig file. The Fig 3.2 format specification tells that the origin is always the upper left corner of the Fig file. This option is turned off by default.
- **verbose output** = *boolean:verbose*
chooses, whether or not to write additional comments to the output file. This option is turned off by default.
- **accept unknown paper size** = *boolean:accept*
chooses, whether or not to accept unknown paper size names. As fig2vect obtains image width and height from inspecting the graphics elements (except the texts) we don't need to worry about unknown paper size names. This option is turned on by default.
- **remove background rectangle** = *boolean:remove*
chooses, whether or not we ignore background rectangles when writing output. A background rectangle is a polygon (type 2, subtype 2), in layer 999, having line width 0, stroke color default (-1) or white – depending on the “background rectangle color” setting – and no filling.
A background rectangle is used to specify the image width and height, either if there are texts near the image border or if additionally space is needed between the outermost graphics elements and the image border.
For compatibility with existing applications this option is turned off by default.

- **background rectangle color** = *string:colordef*
allows to choose the stroke color for a background rectangle. The default setting “default” expects background rectangles to have stroke color -1 (“default”) assigned, the setting “white” changes this to color 7 (white).
This Option may be of interest for jFig users as Edit function to set object properties does not allow to choose -1 (“default”) as stroke color. You can only specify “default” or “white” here, no other colors.

Numeric precision

- **color digits** = *integer:digits*
specifies how many digits after the decimal dot to use in color specifications. Default: 3.
- **position digits** = *integer:digits*
specifies how many digits after the decimal dot to use in co-ordinates specifications. Default: 2.
- **additional trigonometric digits** = *integer:digits*
specifies how many additional digits after the decimal dot to use in co-ordinates if trigonometric calculations were involved. Default: 3, so we have 5 decimal digits after the decimal dot at all.

Spline treatment

- **spline segments** = *integer:segments*
specifies how many Bezier-spline-segments to use per X-spline-segment. Default: 2. Higher values will improve the quality, but more than 8 will not result in visible quality improvement. Figure 1.4 on the next page shows two closed splines (approximated and interpolated) rendered with 2 and 8 Bezier-spline-segments per X-spline-segment.



Figure 1.4: spline segments = 2 / 8

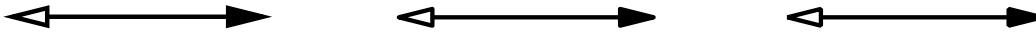


Figure 1.5: arrowhead linejoin = mitered / rounded / beveled

Arrowheads

- **arrowhead linejoin** = *string:linejoin*
chooses how to draw corners of arrowheads. Possible values: “mitered” (default), “rounded” and “beveled”. Figure 1.5 shows the linejoin styles on arrowheads.
- **min iteration steps** = *integer:number*
max iteration steps = *integer:number*
When attaching arrowheads we must shorten the lines a little bit, see section 2.1.4 on page 71. The Newton iteration scheme is used when shortening splines. These two configuration entries set up the minimum and maximum number of iteration steps. If the maximum number of steps is reached without success, the spline is not shortened and the arrowhead is removed (the program prints a warning).

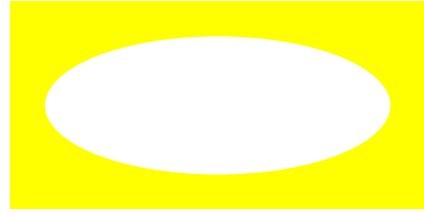
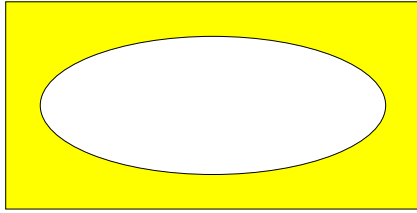


Figure 1.6: remove zero borders = no / yes

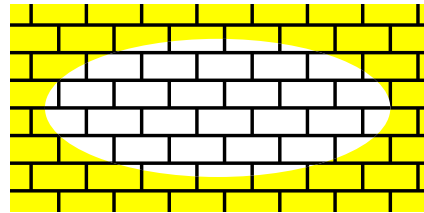
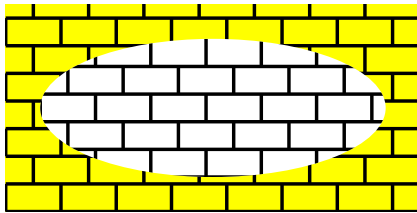


Figure 1.7: fill patterns = yes / contiguous

Fill patterns and dash patterns

- **remove zero border** = *boolean:remove*
Line width 0 allows two interpretations for filled objects:
 - The PostScript interpretation: draw the thinnest possible line. Set the entry to “no”.
 - The common sense interpretation: If the object is filled and the border width is 0, we don’t want the border. Set the entry to “yes”.
- **fill patterns** = *boolean:filling*
chooses, whether or not to allow the use of fill patterns. Specify either “yes”, “no” or “contiguous”. (see figure 1.7).
- **pattern repeat** = *integer:distance*
specifies the distance used to repeat patterns in $\frac{1}{80}$ inch (the same unit as used for line widths). The default value is 4 to show the same pattern distance as the XFig application.
- **dashpattern dot length** = *string:length*
The dots in dash patterns can be drawn with zero length (“0”) or as long as the line width (“linewidth”, default).

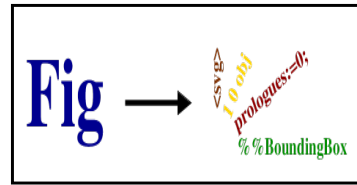
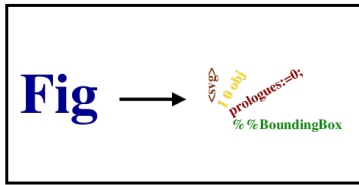


Figure 1.8: keep bitmap aspect ratio = yes / no

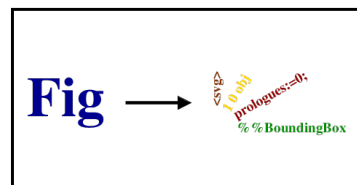
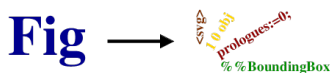


Figure 1.9: remove bitmap border = yes / no

Embedded images

- **keep bitmap aspect ratio** = *boolean:keep*

When drawing an image from file into a box we can

- either keep the width/height relation of the image intact (“yes”, default) leaving some space of the box empty or
- fill the entire box using different scale factors for width and height (“no”).

Figure 1.8 illustrates this option using a Fig file with a PNG-version of the fig2vect-logo embedded. An additional rectangle was placed behind the image box to show the box rectangle.

- **remove bitmap border** = *boolean:remove*

chooses, whether or not to stroke the image box polygon. The default “yes” prevents stroking the polygon. Figure 1.9 illustrates this option. Line width and color of image boxes can not be modified in the “Edit” function of XFig, if you want to draw the box polygon you have to edit the Fig file manually.

- **fill bitmap background** = *boolean:fill*



Figure 1.10: fill bitmap background = no / yes

chooses whether or not the image box is filled using the fill color or fill pattern specified in the Fig file before adding the image. Fill color and/or pattern can not be modified using the XFig “Edit” function, you have to edit the Fig file manually. Figure 1.10 illustrates this option, color “light blue” (11) and enlightenment (pattern 30) are used.

Text typesetting

- **font scale factor** = *float:factor*
specifies a scale factor for texts. Usefull values are in the range $0.9 \leq 0.92 \leq 0.95$. If the default factor 1 is used, texts in output files are larger in relation to other graphics elements than shown in XFig.
- **normal text** = *string:text-handling*
specifies how to handle non-special text. The *text-handling* string consists of key-value pairs, key and value are separated by colon, the pairs in the list are separated by comma. Use the following keys:
 - handling
chooses whether text is handled by the driver itself (“none”) or by $\text{T}_{\text{E}}\text{X}/\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ (“tex”).
 - font
chooses how to find the font to use:
 - * “fig” requires exactly the font specified in the Fig file.
 - * “similar” can be used for automatic font replacement (A font set up by the $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ preamble is choosen, having the same features – serif/sans-serif/monospaced, normal/bold, upright/italic – as the font specified in the Fig file).

- * "tex" leaves all font selection up to \LaTeX .
- size
chooses, how to calculate the font size, either as specified in the Fig file ("fig") or leave the choice up to \LaTeX ("tex").
- mbox
This key does not have a value. If the key is found, the text is encapsulated in a `\mbox` instruction.
- **special text** = *string:text-handling*
specifies how to handle special text. The *text-handling* contains the same settings as for "normal text", except that "handling" is ignored (special text is always handled by \TeX/\LaTeX).
- **tex command** = *string:tex-command*
chooses whether to use \LaTeX ("latex", default) or plain \TeX ("tex").
- **latex font setup** = *string:setup-name*
chooses a predefined \LaTeX preamble for font setup. Use "pdf", "ams", "newcent", "pdf12", "ams12" or "newcent12".
Note: The preferred method to establish a \LaTeX preamble for font setup is the "preamble file" configuration entry.
- **preamble file** = *file-name*
specifies the file name for a stripped-down preamble for font setup matching the font setup in your \LaTeX documents.
This preamble file must not contain "`\begin{document}`".
- **repeat error messages** = *boolean:repeat*
If a driver can not do \LaTeX text typesetting we can show the warning for each text ("yes", default) or only once ("no").
- **skip all texts** = *yes*
If we use the combination of EPS/ \TeX or PDF/ \TeX driver we do not want any text-handling related warning from the EPS or PDF driver at all. The value ("yes") allows to skip all these error messages.
- **utf-8** = *boolean*
turns UTF-8 support on or off. The special value "auto" can be specified to inspect the LANG environment variable whether or not it ends on ".utf-8".
If UTF-8 support is turned on `fig2vct` UTF-8 decodes all *non-special* texts.

Processing of Unicode characters depends on the output driver and the text handling specified for normal text. If \TeX/\LaTeX is used for text handling (“handling:tex”) `fig2vect` attempts to find and write \LaTeX code to represent the Unicode character.

If normal text is handled directly by the driver (“handling:none”) Unicode characters in the range `0x00000000...0x000000FF` are printed directly, all other characters are ignored.



Figure 1.11: use metapost arrowheads = yes / no

1.5.2 Configuration entries for the MetaPost driver

Arrowheads

- **use metapost arrowheads** = *boolean:mp-arrowheads*
allows or denies the use of MetaPost arrowheads. MetaPost adds curved arrowheads to curved lines (arcs, splines) but does not support all the arrowhead types defined in Fig. Some Fig arrowhead types are converted to other types when using MetaPost arrowheads, (see figure 1.11).

Text typesetting

- **normal text** = *string:text-handling*
 - **special text** = *string:text-handling*
- handling
- * **handling:tex**
converts texts into "`\btex ... \etex`".
 - **font:fig**
Use the \LaTeX font corresponding exactly to the PostScript-Font specified in the Fig file.
 - **font:similar**
Select a \LaTeX font by features.
 - **font:tex**
Do not explicitly request a font, let \LaTeX choose.
 - * **handling:none**
converts texts into "`label "... "`".
 - **font:fig**
The PostScript font name is used. "`label "... "` infont "Times-Roman"".

- `font:tex`

The \LaTeX font name is used. "`label "... infont "ptmr"`".

Note: For "`handling:none`" it is recommended to use "`font:fig`", the use of \LaTeX font names may lead to problems.

- **embed fonts** = *boolean:embedding*

chooses whether or not the `mpost` (`mp`) program embeds the fonts into the output file. The "`mp`" driver adds either "`prologues:=0;`" or "`prologues:=1;`" to the MetaPost source. Embedding the fonts is only useful to produce standalone images. If you want to use the image in \LaTeX documents you should not embed the fonts to each image, leave font embedding up to `dvips` or `pdf \LaTeX` , these programs are specialists in it.

1.5.3 Configuration entries for the EPS/PS driver

PS level and DSC

- **ps level** = *integer:level*
chooses a PostScript level, either “1”, “2” or “3”.
- **dsc comments** = *integer:level*
chooses the PostScript level for DSC comments. DSC comments are used for document management, i.e. to extract pages or page ranges for printing or to keep fonts and procedures locally on the printer instead of downloading them with each print job. Sometimes document management is separated from PostScript interpretation, i.e. in network print systems one must choose a PostScript level understood by the printer and DSC level understood by the document management software on the print queue scheduler. Choose “1”, “2”, “3”, “yes” (use same level as ps level) or “no” (do not print DSC comments) here.
- **ps showpage** = *boolean:show-page*
chooses, whether or not to add a showpage operator to the output file. This operator is recommended for standalone images. Images which are included in documents should not contain this operator. This operator can only be used in *.ps files, not in *.eps files.
- **page size** = *integer:w integer:h [integer:llx integer:lly integer:urx integer:ury]*
sets paper width and height and optionally the printable range. This option is only available when producing PS, not EPS.
- **page alignment** = *string:alignment*
chooses the position of the graphics within the printable range. Use “top” or “bottom” to change the vertical position or “left” or “right” to change the horizontal position.
By default the graphics is placed centered both horizontally and vertically. This option is only available when producing PS, not EPS.
- **ps setpagedevice** = *boolean:set-page-size*
controls, whether or not the setpagedevice operator is used to set the page size (“paper size”) to match the bounding box. This operator is recommended for standalone images which are processed by GhostScript.

Do not use this operator for images which are included in documents. The operator can only be used in *.ps files, not in *.eps files.

Virtual memory

- **bitmap image dictionary** = *boolean:use-dict*

To handle embedded images EPS images define strings using the “def” operator. Additionally there is a definition for a filtered input stream. Setting this configuration entry to

```
1 bitmap image dictionary = yes
```

the EPS drawing instructions are embedded into a

```
1 ... dict begin
2 % Zeichenbefehle
3 end
```

construction.

The final “end” keyword removes the dictionary from the dictionary stack, there is no longer any reference to the dictionary, it is unusable and the memory can be released by the garbage collection.

The only references to the strings and to the filter were in the dictionary, so the strings are no longer accessible and can also be removed by the garbage collection.

Note: Do not use this configuration option if the Fig file includes PS/EPS images. The dictionary size is calculated using the number of different image width. PS/EPS images might contain other definitions, so the dictionary size is insufficient.

- **force garbage collection** = *boolean:gc*
chooses whether or not to ask for the garbage collection using “1 vmreclaim” at the end of the output. This is only usefull if

```
1 bitmap image dictionary = yes
```

was used.

Encoding of embedded bitmap images

- **ps run-length encoding** = *boolean:allow*
allows or denies the use of run-length encoding.
- **separated rgb channels** = *boolean:separate*
allows or denies the use of separated channels for red, green and blue data. If the separated channels are used, the EPS file contains first all red-values, followed by all green-values and all blue-values. This allows a good run-length compression of areas in the same color. Without the separated channels, the EPS file contains the red-value, the green-value and the blue-value for the first pixel. . . In this case, run-length compression can only compress grey areas. The disadvantage of separated channels is the higher need for virtual memory in the PostScript interpreter. For separated channels we need $M_s = 3 \cdot w \cdot h$ bytes for each bitmap, without separated channels we need $M_n = w$ bytes (w is the image width in pixels, h the image height).

1.5.4 Configuration entries for the PDF driver

General options

- **plain text streams** = *boolean:plain*
chooses whether to write graphics instructions into a plain text stream or into a compressed stream (default).
- **full screen** = *boolean:full-screen*
chooses whether or not to switch to full text mode when opening the PDF file.
- **arc bezier steps** = *integer:steps*
specifies the number of Bezier-spline-segments to use for each quadrant of a circle (default: 2).
- **allow pdf page attributes** = *boolean:attributes*
enables/disables the creation of PDF page attributes.
PDF page attributes are needed if the Fig file references PNG files containing an alpha channel if the PDF output file is intended for standalone viewing.

Fill patterns and dash patterns

- **tiled patterns** = *boolean:tiled*
chooses whether or not to use pattern tiles for fill patterns. If tiled patterns are used and the PDF file is viewed on screen there are sometimes irritations on tile borders. The large (non-tiled) pattern cells result in better screen rendering but increase the file size of the PDF file.

Embedded images

- **interpolate images** = *boolean:interpolate*
activates or deactivates image interpolation for embedded bitmaps.
- **flip direction** = *string:flip-dir*
chooses the image flip direction, either “diagonal” (default) or “horizontal” (like seen in jFig).

1.5.5 Configuration entries for the TeX driver

- **full tex file** = *boolean:full-flag*
specifies whether or not to produce a full T_EX file to produce a standalone PDF image. By default we produce code snippets to be included into L^AT_EX documents.

1.5.6 Configuration entries for the SVG driver

SVG header and SVG structure

- **svg version** = *string:version*
chooses the SVG version, either “1.0” (default), “1.1” or “1.2”. “1.0” is recommended as I do not know any browser plugin supporting a newer version.
- **embedded svg** = *boolean:embedded*
chooses whether to produce an SVG snippet for inclusion into a XML document or to produce a standalone SVG file (default).
- **wh specification** = *string:wh-spec*
chooses how to specify width and height in the header, either “inches” (default), “pixels” or “points” (PostScript-Points).

Object attributes

The SVG file format allows different methods to specify object attributes:

- multiple single attributes, i.e.:

```
1 <rect fill="yellow" stroke="black" stroke-width="0.9" .../ >
```

- one style attribute, i.e.:

```
1 <rect style="fill: yellow; stroke: black; stroke-width: 0.9;" .../ >
```

- use of classes and CSS style sheets:

```
1 <defs><style type="text/css"><![CDATA[
2   .c1 {
3     fill: yellow;
4     stroke: black;
5     stroke-width: 0.9;
6   }
7 ]]></style></defs>
8 <rect class="c1" .../ >
```

The use of classes and style sheets may decrease the file size significantly if the images contains lots of objects with equal attributes.

Classes and style attributes have higher priority than the multiple single attributes. If you want to change the single attributes in animations or event handlers you should avoid the use of classes and the use of the style attribute.

- **prepare for modifications** = *boolean:mods*
requires the usage of multiple single attributes. If fig2vect finds ECMA script, this mode is turned on automatically.
- **use css** = *boolean:use-css*
controls whether to use CSS style sheets or to use the style attribute. Only of relevance if the option above is turned off and the Fig file does not contain ECMA script.

Fonts

- **gs svg-font directory** = *string:directory*
configures the driver to look for SVG fonts in the specified directory. This option is deprecated, you should use a font configuration file instead (see below).
- **font configuration file** = *string:filename*
specifies the file name of a font configuration file, i.e. “winfont.cfg”. The configuration file search mechanism provided by the dklibs package is used to find these configuration files. I recommend to place font configuration files in the directory `${prefix}/etc/fig2vect` or `%WINDIR%\fig2vect` to make sure they are found. The file structure is described in dkfont(5) and in the dklibs manual.

ECMA Script

- **js library** = *string:filename*
uses the specified file as library containing ECMA script functions.

To use ECMA script some object need an ID. Control comments can be used to assign the IDs, i.e.:

```
1 svg: id = obj001
```

The control comment must be assigned to object which is the holder of the ID. Control comments can also be used to assign event handlers, i.e.:

```
1 svg: onmouseover = changebgcolor(evt)
```

There is support for the “onfocusin”, “onfocusout”, “onactivate”, “onclick”, “onmousedown”, “onmouseup”, “onmouseover”, “onmousemove”, “onmouseout”, “onload”, “onunload”, “onabort”, “onerror”, “onresize”, “onscroll”, “onzoom”, “onbegin”, “onend” and “onrepeat”.

Handlers for document events (i.e. onload) can be assigned using control comments in the document comment range – between line 8 (transparent color) and 9 (resolution/origin). The file doc2.fig⁷ is an example for a Fig file containing control comments for ECMA script.

⁷<http://fig2vect.sourceforge.net/doc2.fig>

1.6 Additional hints

1.6.1 Background rectangles

The `fig2vect` program obtains the image width and height from an inspection of all graphics elements (except texts). You might want to use a background rectangle for two reasons:

- There are texts on outermost positions of the image or
- you want to add additional space between the outermost objects and the image border.

A background rectangle is a polygon (type 2), subtype box (2) in layer 999 having a line width 0, line color -1 (default) and fill style -1 (no filling). In XFig you can choose the line color -1 by clicking the “Default” color. The jFig program does not allow to set the line color to “Default”, set the line color to white and use the configuration entry

```
1 background rectangle color = white
```

to configure `fig2vect` to recognize the background rectangle correctly.

The background rectangle must include all other graphics elements completely.

If the option “remove background rectangle” is used, the rectangle is not drawn to the output file, it is only used to calculate the image dimensions.

1.6.2 How to use fill patterns

The disk space and the computation load for pattern filling depends on the driver. Some drivers only need graphics instructions for one pattern tile, other drivers require all the graphics instructions to fill the object.

MetaPost files contain all graphics instructions to fill an object. The disk space requirements for filled objects grows both with object width and height, except for simple patterns (lines and crosshatches). For simple patterns the disk space grows with the summary of width and height. If you have a choice you should prefer the use of simple patterns.

EPS uses procedures to draw the fill patterns. The disk space does not depend on object width and height.

PDF can use either small (tiled) pattern cells or large (non-tiled) pattern cells. When using small pattern cells object fillings are created by a sequence of small cells. Mathematical roundings may result in irritations on cell borders when rendering the PDF file on screen. So we can also use a large pattern cell (a pattern cell as large as the entire PDF file). As there are no pattern cell borders there are no irritations at pattern cell borders. As the large pattern cell contains all graphic instructions to draw the fill pattern on an area as large as the PDF file significantly more disk space is needed. Like for the MetaPost driver you should prefer simple patterns (lines and crosshatches) when using large pattern cells.

SVG files use small pattern cells to create pattern fillings.

1.6.3 Embedded images

The jFig and XFig programs behave differently when embedding images. For jFig it does not matter, which polygon corner is specified first.

XFig rotates the embedded image if you do not specify the upper left corner first. The rotation is in multiples of 90°.

The fig2vect program does not care about these rotations, it uses the polygon points to find the left and right x -border and upper and lower y -border. No rotation is applied to embedded images.

If you want a rotation, edit the image in an image editor, do the rotation and save the file. Embed the file in XFig without additional rotation.

1.6.4 SVG images

SVG images can be used for two purposes:

- publishing graphics on web sites and
- embedding images into documents.

Fig files use the 35 PostScript fonts, these fonts are not available in SVG by default. You must ever provide fonts to the SVG viewer or substitute the fonts.

For web graphics it is usefull to provide SVG variants of the Ghostscript fonts for download and reference the downloadable fonts in the SVG images.

To embed SVG images – i.e. in a Windows text processing application – you should substitute the fonts. The SVG file should reference those fonts used in the document text.

Configuration files can be used for both purposes, the configuration file name is specified either using the “-o font.configuration.file=*filename*” command line option or using the “font configuration file = *filename*” configuration entry.

Two example configuration files are shipped with fig2vect:

- webfont.cfg
shows how to reference downloadable SVG fonts and
- winfont.cfg
shows how to reference system fonts.

Both files need customizing before they can be used.

In “webfont.cfg” correct the “directory” entry for each font, i.e. use “<http://www.my-corp.com/fonts-urw-svg>”.

In “winfont.cfg” make sure to reference only fonts and font families wich are available on you system. Use “fonts” in the control panel to view the collection of available fonts.

1.7 Bitmap images

1.7.1 Overview

The `fig2vect` program can not create bitmap images directly. Instead you create a vector graphics file using `fig2vect` and convert the vector graphics file into a bitmap image using a conversion program (GhostScript or Batik rasterizer).

The following vector graphics file formats can be used:

- EPS/PS

The EPS file is converted to PNG using GhostScript.

You should not use EPS as intermediate format if there are transparent PNG images embedded into the Fig file or if the Fig file contains special text.

GhostScript seems to ignore the BoundingBox information in the EPS file and creates an output image corresponding to a printer page. Manual post-processing of the PNG file is necessary to cut the image. Use the configuration entry “`ps setpagedevice = yes`” or the command line option “`-o ps.setpagedevice=yes`” for the PS driver (“`-lps`”) to avoid this problem.

Alternatively you can provide the “`-dEPSCrop`” option to GhostScript to set output file image size corresponding to the EPS BoundingBox.

- PDF

Use of PDF as intermediate format is the preferred way to create bitmap images. The PDF file is converted to PNG using GhostScript. Embedded transparent PNG files or special texts in the Fig file can be processed without problems.

Unfortunately there are problems with some versions of GhostScript. GhostScript 7.07 on Fedora Core 3 died with “segmentation fault” on my Laptop, but GhostScript 8.15 on Windows XP Prof. processed the file without problems.

- SVG

The Batik rasterizer can be used to convert SVG files to PNG. All image areas not touched by graphics operations in the Fig files are left transparent in the PNG output. So this is a good way to create partially transparent images, i.e. for Web logos.

For best conversion results you should have the SVG ports of the GhostScript fonts installed, see 1.2.3 on page 11.

Conversion from Fig to SVG is only possible if the Fig file does not contain special text.

1.7.2 Detailed conversion steps

Bitmap images via EPS

Use

```
fig2vect -leps.standalone demofig.fig demofig.eps
```

to convert the Fig file to EPS and

```
gs options -sOutputFile=demoeps.png demofig.eps
```

to convert the EPS file to PNG. The options are recommended as in table 1.1. On Windows systems use “gswin32c” instead of “gs”.

Table 1.1: GhostScript options for conversion to PNG

Option	Meaning
-dSAFER	switches gs into safe mode and restricts activities possibly doing harms to your computer.
-dBATCH	runs gs in batch mode, gs exits after processing the input file instead of reading further commands from standard input.
-dNOPAUSE	disables the prompt for ENTER after each page.
-dEPSCrop	chooses output image size to fit the EPS BoundingBox.
-dNOCACHE	disables glyph bitmap caching. The manual recommends this option for debugging purposes only but there were articles in newsgroups telling that this option increases output quality sometimes.
-sDEVICE=png16m	choose output file format PNG.
-r200	sets resolution to 200dpi (you may choose a different resolution)
<i>to be continued</i>	

<i>Continuation</i>	
<code>-dGraphicsAlphaBits=4</code> <code>-dTextAlphaBits=4</code>	sets options for edge-smoothing (german: Kantenglättung). You can use 1, 2, 4... and powers of 2 or omit these options.

Bitmap images via PDF

Run

```
fig2vect -l pdf demofig.fig demofig.pdf
```

to create the PDF file.

If the Fig file contains text, use

```
fig2vect -lpdf.tex demofig.fig demoi.pdf
```

```
fig2vect -ltex.full -idemoi.pdf demofig.fig
```

demofig.tex

```
pdflatex demofig && pdflatex demofig
```

```
gs options -sOutputFile=demopdf.png demofig.pdf
```

instead. For `gs` use the same options as in the previous section (see table 1.1 on page 56).

Bitmap images via SVG

Run

```
fig2vect -lsvg demofig.fig demofig.svg
```

to convert the Fig file to SVG. You may add the option “-o wh.specification=pixels” if you want to change image width and height manually before converting to PNG

Use

```
java -jar ../../batik-rasterizer.jar demofig.svg
```

to convert the SVG to PNG (demofig.png).

Chapter 2

Technical details

2.1 Mathematical aspects of the fig2vect program

2.1.1 Before you read this section

I am neither a native english speaker nor a mathematician. So my knowledge of english words and phrases in the field of mathematics is poor. If you think this text needs corrections, please do not simply complain, send a corrected version of the text instead.

2.1.2 Curves, X-splines and Bezier-splines

Curves

Curves in \mathbb{R}^2 can be described by the equations:

$$x = x(t) \quad y = y(t) \quad t \in [t_S; t_E]$$

In computer graphics a range $t \in [0; 1]$ is oftenly used either for an entire curve or a curve segment

A spline is a description of a curve, a set of discrete points (control points) is used to describe a contiguos curve.

Cubic Bezier-Splines

A cubic Bezier spline is defined by the final points $P_i(x_i, y_i)$ and $P_j(x_j, y_j)$ and additional control points $P_{ir}(x_{ir}, y_{ir})$ and $P_{jl}(x_{jl}, y_{jl})$, $j = i + 1$. P_{ir} is the control

point on the “right” side of P_i . P_{jl} is the control point on the “left” side of P_j . Sometimes the notation P_{i+} and P_{j-} or P_i^+ and P_j^- is used for the control points.

$$\begin{aligned}x(t) &= x_i(1-t)^3 + 3x_{ir}(t-1)^2t + 3x_{jl}(1-t)t^2 + x_jt^3 \\y(t) &= y_i(1-t)^3 + 3y_{ir}(t-1)^2t + 3y_{jl}(1-t)t^2 + y_jt^3 \\t &\in [0; 1] \\x(t) &= x_i(1-2t+t^2)(1-t) + 3x_{ir}(1-2t+t^2)t + 3x_{jl}(1-t)t^2 + x_jt^3 \\&= x_i(1-3t+3t^2-t^3) + 3x_{ir}(t-2t^2+t^3) + 3x_{jl}(t^2-t^3) + x_jt^3 \\x'(t) &= x_i(-3+6t-3t^2) + 3x_{ir}(1-4t+3t^2) + 3x_{jl}(2t-3t^2) + x_j \cdot 3t^2\end{aligned}$$

If x_i, x'_i, x'_j and x_j are given, the control points can be calculated using:

$$\begin{aligned}x_{ir} &= x_i + \frac{1}{3}x'_i(0) \\x_{jl} &= x_j - \frac{1}{3}x'_j(1)\end{aligned}$$

For y :

$$\begin{aligned}y_{ir} &= y_i + \frac{1}{3}y'_i(0) \\y_{jl} &= y_j - \frac{1}{3}y'_j(1)\end{aligned}$$

For each internal curve point there are two additional control points.

X-Splines

Cross-splines (X-splines) were introduced by Schlick and Blanc in 1995. They are not explained in detail here, see the SIGGRAPH'95 paper. This section is focused on the usage of X-splines in Fig-files and processing of X-splines in fig2vect.

A function $f(u)$ is used as blending function for approximations:

$$f(u, p) = \begin{cases} u^3 \left(10 - p + (2p - 15)u + (6 - p)u^2 \right) & \text{for } u \in [0; 1] \\ 0 & \text{otherwise} \end{cases}$$

$$= \begin{cases} (6 - p)u^5 + (2p - 15)u^4 + (10 - p)u^3 & \text{for } u \in [0; 1] \\ 0 & \text{otherwise} \end{cases}$$

Two functions are used to build the blending function $e(u)$ for interpolations:

$$e(u, p, q) = \begin{cases} qu + 2qu^2 + (10 - 12q - p)u^3 \\ \quad + (2p + 14q - 15)u^4 + (6 - 5q - p)u^5 & \text{for } u \in [0; 1] \\ qu + 2qu^2 - 2qu^4 - qu^5 & \text{for } u \in [-1; 0) \\ 0 & \text{otherwise} \end{cases}$$

Deriving the functions results in:

$$\frac{\partial f}{\partial u} = \begin{cases} 5(6 - p)u^4 + 4(2p - 15)u^3 + 3(10 - p)u^2 & \text{for } u \in [0; 1] \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial e}{\partial u} = \begin{cases} 5(6 - 5q - p)u^4 + 4(2p + 14q - 15)u^3 \\ \quad + 3(10 - 12q - p)u^2 + 4qu + q & \text{for } u \in [0; 1] \\ q + 4qu - 8qu^3 - 5qu^4 & \text{for } u \in [-1; 0) \\ 0 & \text{otherwise} \end{cases}$$

When processing Fig files we have equidistant control points, this means $\Delta t = 1$.

An X-spline is described by n control points ($0 \leq i \leq n - 1$). An additional parameter s_k ($-1 \leq s_k \leq 1$) is assigned to each control point P_k at $t = T_k$. P_k is an approximation point if $0 \leq s_k \leq 1$ or an interpolation point if $-1 \leq s_k < 0$.

Each control point produces a left-side blending function F_k^- and a right-side blending function F_k^+ . The blending functions control the impact of the control point on the curve in the environment around the control point. Each of the control functions can have an impact on up to two segments, a control point has an impact on up to 4 segments.

To choose the left-side and right-side blending function we need to take care of the control points in the neighbourhood too, see table 2.1.

Table 2.1: Choosing the blending functions

Control point type		Blending function	
P_i	P_j	F_i^+	F_j^-
Approx.	Approx.	$T_i^+ = T_j + s_j \Delta t$ $p_i^+ = \frac{2(T_i^+ - T_i)^2}{\Delta t^2}$ $F_i^+ = f\left(\frac{t - T_i^+}{T_i - T_i^+}, p_i^+\right)$	$T_j^- = T_i - s_i \Delta t$ $p_j^- = \frac{2(T_j^- - T_j)^2}{\Delta t^2}$ $F_j^- = f\left(\frac{t - T_j^-}{T_j - T_j^-}, p_j^-\right)$
Approx.	Interp.	$T_i^+ = T_j$ $p_i^+ = \frac{2(T_i^+ - T_i)^2}{\Delta t^2}$ $F_i^+ = f\left(\frac{t - T_i^+}{T_i - T_i^+}, p_i^+\right)$	$T_j^- = T_i - s_i \Delta t$ $p_j^- = \frac{2(T_j^- - T_j)^2}{\Delta t^2}$ $F_j^- = f\left(\frac{t - T_j^-}{T_j - T_j^-}, p_j^-\right)$
Interp.	Approx.	$T_i^+ = T_j + s_j \Delta t$ $p_i^+ = \frac{2(T_i^+ - T_i)^2}{\Delta t^2}$ $F_i^+ = f\left(\frac{t - T_i^+}{T_i - T_i^+}, p_i^+\right)$	$T_j^- = T_i$ $p_j^- = \frac{2(T_j^- - T_j)^2}{\Delta t^2}$ $F_j^- = f\left(\frac{t - T_j^-}{T_j - T_j^-}, p_j^-\right)$
<i>to be continued...</i>			

<i>Continuation</i>			
Interp.	Interp.	$T_i^+ = T_i + 2\Delta t$ $p_i^+ = 2$ $q_i^+ = -\frac{1}{2} \cdot s_i$ $F_i^+ = e\left(\frac{T_j - t}{\Delta t}, p_i^+, q_i^+\right)$	$T_j^- = T_j - 2\Delta t$ $p_j^- = 2$ $q_j^- = -\frac{1}{2} \cdot s_j$ $F_j^- = e\left(\frac{t - T_i}{\Delta t}, p_j^-, q_j^-\right)$
Approx., P_i is left final point of open spline	Interp.	$T_i^+ = T_i + 2\Delta t$ $p_i^+ = 2$ $q_i^+ = -\frac{1}{2} \cdot s_j$ $F_i^+ = e\left(\frac{T_j - t}{\Delta t}, p_i^+, q_i^+\right)$	$T_j^- = T_i$ $p_j^- = \frac{2(T_j^- - T_j)^2}{\Delta t^2}$ $F_j^- = f\left(\frac{t - T_j^-}{T_j - T_j^-}, p_j^-\right)$
Interp.	Approx., P_j is right final point of open spline	$T_i^+ = T_j$ $p_i^+ = \frac{2(T_i^+ - T_i)^2}{\Delta t^2}$ $F_i^+ = f\left(\frac{t - T_i^+}{T_i - T_i^+}, p_i^+\right)$	$T_j^- = T_j - 2\Delta t$ $p_j^- = 2$ $q_j^- = -\frac{1}{2} \cdot s_i$ $F_j^- = e\left(\frac{t - T_i}{\Delta t}, p_j^-, q_j^-\right)$

By setting $\Delta t = 1$ we can use simplifications as shown in table 2.2 on the next page.

Table 2.2: Choosing the blending functions in fig2vect

Control point type		Blending function	
P_i	P_j	F_i^+	F_j^-
Approx.	Approx.	$T_i^+ = j + s_j$ $p_i^+ = 2(T_i^+ - i)^2$ $F_i^+ = f\left(\frac{t-T_i^+}{i-T_i^+}, p_i^+\right)$	$T_j^- = i - s_i$ $p_j^- = 2(T_j^- - j)^2$ $F_j^- = f\left(\frac{t-T_j^-}{j-T_j^-}, p_j^-\right)$
Approx.	Interp.	$T_i^+ = j$ $p_i^+ = 2$ $F_i^+ = f\left((j-t), p_i^+\right)$	$T_j^- = i - s_i$ $p_j^- = 2(T_j^- - j)^2$ $F_j^- = f\left(\frac{t-T_j^-}{j-T_j^-}, p_j^-\right)$
Interp.	Approx.	$T_i^+ = j + s_j$ $p_i^+ = 2(T_i^+ - i)^2$ $F_i^+ = f\left(\frac{t-T_i^+}{i-T_i^+}, p_i^+\right)$	$T_j^- = i$ $p_j^- = 2$ $F_j^- = f\left((t-i), p_j^-\right)$
Interp.	Interp.	$T_i^+ = i + 2$ $p_i^+ = 2$ $q_i^+ = -\frac{1}{2} \cdot s_i$ $F_i^+ = e\left((j-t), p_i^+, q_i^+\right)$	$T_j^- = j - 2$ $p_j^- = 2$ $q_j^- = -\frac{1}{2} \cdot s_j$ $F_j^- = e\left((t-i), p_j^-, q_j^-\right)$
to be continued...			

<i>Continuation</i>			
Approx., P_i is left final point of open spline	Interp.	$T_i^+ = i + 2$ $p_i^+ = 2$ $q_i^+ = -\frac{1}{2} \cdot s_j$ $F_i^+ = e\left((j-t), p_i^+, q_i^+\right)$	$T_j^- = i$ $p_j^- = 2$ $F_j^- = f\left((t-i), p_j^-\right)$
Interp.	Approx., P_j is right final point of open spline	$T_i^+ = j$ $p_i^+ = 2$ $F_i^+ = f\left((j-t), p_i^+\right)$	$T_j^- = j - 2$ $p_j^- = 2$ $q_j^- = -\frac{1}{2} \cdot s_i$ $F_j^- = e\left((t-i), p_j^-, q_j^-\right)$

After choosing blending functions F_k^- and F_k^+ for all control points P_k we can calculate curve points using

$$\begin{aligned}
 z(t) &= x_{i-1}F_{i-1}^+(t) + x_iF_i^+(t) + x_jF_j^-(t) + x_{j+1}F_{j+1}^-(t) \\
 v(t) &= F_{i-1}^+(t) + F_i^+(t) + F_j^-(t) + F_{j+1}^-(t) \\
 z'(t) &= x_{i-1} \frac{\partial F_{i-1}^+(t)}{\partial t} + x_i \frac{\partial F_i^+(t)}{\partial t} + x_j \frac{\partial F_j^-(t)}{\partial t} + x_{j+1} \frac{\partial F_{j+1}^-(t)}{\partial t} \\
 v'(t) &= \frac{\partial F_{i-1}^+(t)}{\partial t} + \frac{\partial F_i^+(t)}{\partial t} + \frac{\partial F_j^-(t)}{\partial t} + \frac{\partial F_{j+1}^-(t)}{\partial t} \\
 x(t) &= \frac{z(t)}{v(t)} \\
 x'(t) &= \frac{v(t) \cdot z'(t) - z(t) \cdot v'(t)}{v^2(t)}
 \end{aligned}$$

The calculations for $y(t)$ and $y'(t)$ are similar.

To simplify the calculation function in the program we place each spline segment into four points A , B , C and D . B is the point for $t = 0$. C is the point for $t = 1$. A is the left neighbour of $t = 0$, D is the right neighbour of $t = 1$.

This means:

$$T_A = -1 \quad T_B = 0 \quad T_C = 1 \quad T_D = 2$$

To calculate a point between B and C we need the functions F_A^+ , F_B^+ , F_C^- and F_D^- , to calculate the derivation we need $\frac{\partial}{\partial t}F_A^+$, $\frac{\partial}{\partial t}F_B^+$, $\frac{\partial}{\partial t}F_C^-$ and $\frac{\partial}{\partial t}F_D^-$.

In the beginning we assume that all points are approximation points:

$$\begin{aligned} T_A^+ &= s_B & p_A^+ &= 2(-1 - T_A^+)^2 & u_A(t) &= \frac{t - T_A^+}{-1 - T_A^+} & u'_A(t) &= -\frac{1}{T_A^+ + 1} \\ F_A^+(t) &= f(u_A(t), p_A^+) \\ T_B^+ &= 1 + s_C & p_B^+ &= 2T_B^{+2} & u_B(t) &= \frac{t - T_B^+}{-T_B^+} & u'_B(t) &= -\frac{1}{T_B^+} \\ F_B^+(t) &= f(u_B(t), p_B^+) \\ T_C^- &= -s_B & p_C^- &= 2(1 - T_C^-)^2 & u_C(t) &= \frac{t - T_C^-}{1 - T_C^-} & u'_C(t) &= \frac{1}{1 - T_C^-} \\ F_C^-(t) &= f(u_C(t), p_C^-) \\ T_D^- &= 1 - s_C & p_D^- &= 2(2 - T_D^-)^2 & u_D(t) &= \frac{t - T_D^-}{2 - T_D^-} & u'_D(t) &= -\frac{1}{2 - T_D^-} \\ F_D^-(t) &= f(u_D(t), p_D^-) \end{aligned}$$

If $s_A < 0$ and $s_B < 0$, we change:

$$\begin{aligned} u_A(t) &= -t & u'_A(t) &= -1 & q_A &= -\frac{1}{2}s_A & p_A^+ &= 2 \\ F_A^+(t) &= e(u_A(t), p_A^+, q_A) & T_A^+ &= 1 \end{aligned}$$

If $s_B < 0$ and $s_C < 0$, we change:

$$\begin{aligned} u_B(t) &= -1 - t & u'_B(t) &= -1 & q_B &= -\frac{1}{2}s_B & p_B^+ &= 2 \\ F_B^+(t) &= e(u_B(t), p_B^+, q_B) & T_B^+ &= 2 \\ u_C(t) &= t & u'_C(t) &= 1 & q_C &= -\frac{1}{2}s_C & p_C^- &= 2 \\ F_C^-(t) &= e(u_C(t), p_C^-, q_C) & T_C^- &= -1 \end{aligned}$$

If $s_C < 0$ and $s_D < 0$, we change:

$$\begin{aligned} u_D(t) &= t - 1 & u'_D(t) &= 1 & q_D &= -\frac{1}{2}s_D & p_D^- &= 2 \\ F_D^-(t) &= e(u_D(t), p_D^-, q_D) & T_D^- &= 0 \end{aligned}$$

For open splines we need further tests:

- For the first spline segment:

If $s_B = 0$ and $s_C < 0$ we change:

$$\begin{aligned} u_B(t) &= 1 - t & u'_B(t) &= -1 & q_B &= -\frac{1}{2}s_C & p_B^+ &= 2 \\ F_B^+(t) &= e(u_B(t), p_B^+, q_B) & T_B^+ &= 2 \\ u_C(t) &= t & u'_C(t) &= 1 & p_C^- &= 2 \\ F_C^-(t) &= f(u_C(t), p_C^-) & T_C^- &= 0 \end{aligned}$$

- For the second segment:

If $s_A = 0$ and $s_B < 0$, we change:

$$\begin{aligned} u_A(t) &= -t & u'_A(t) &= -1 & q_A &= -\frac{1}{2}s_B & p_A^+ &= 2 \\ F_A^+(t) &= e(u_A(t), p_A^+, q_A) & T_A^+ &= 1 \end{aligned}$$

- For the second last segment:

If $s_C < 0$ and $s_D = 0$, we change:

$$\begin{aligned} u_D(t) &= t - 1 & u'_D(t) &= 1 & q_D &= -\frac{1}{2}s_C & p_D^- &= 2 \\ F_D^-(t) &= e(u_D(t), p_D^-, q_D) \end{aligned}$$

- For the last segment:

If $s_B < 0$ and $s_C = 0$, we change:

$$\begin{aligned} u_B(t) &= 1 - t & u'_B(t) &= -1 & p_B^+ &= 2 \\ F_B^+(t) &= f(u_B(t), p_B^+) & T_B^+ &= 1 \\ u_C(t) &= t & u'_C(t) &= 1 & q_C &= -\frac{1}{2}s_B & p_C^- &= 2 \\ F_C^-(t) &= e(u_C(t), p_C^-, q_C) & T_C^- &= -1 \end{aligned}$$

Approximation of X-splines by Bezier splines

The output formats planned for `fig2vect` support cubic Bezier splines. So we need to approximate each X-spline-segment by a sequence of Bezier splines.

First we calculate curve points corresponding to the control points ($t = T_1, t = T_2 \dots$).

To calculate control points for the Bezier splines we need the derivations. X-splines allow corner points, so we need a “left-side” and a “right-side” derivation.

In the next step we calculate curve points and derivations between the control points.

If each X-spline segment is approximated by a sequence of Bezier spline segments, the derivations must be corrected by dividing them by the number m of Bezier spline segments per X-spline segment. This is because the X-spline segment uses $t \in [0; 1]$ for the entire X-spline-segment but Bezier spline segments have $t \in [0; 1]$ for the Bezier spline segment which corresponds to only a part of the X-spline segment.

2.1.3 Arcs

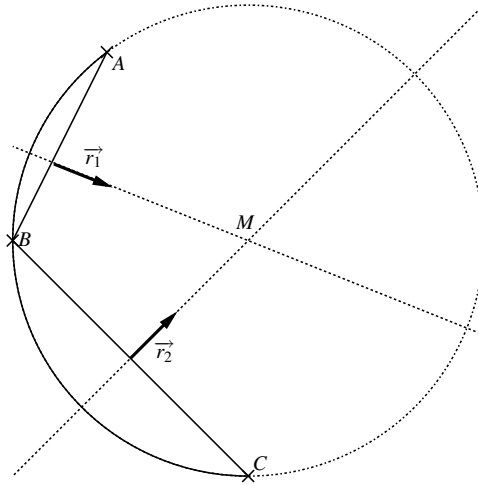


Figure 2.1: Calculations on arcs

Arcs in Fig files are specified by three points $A(x_A, y_A)$, $B(x_B, y_B)$ and $C(x_C, y_C)$.

The notation \vec{r}_A , \vec{r}_B and \vec{r}_C is used for the vectors from $(0;0)$ to A , B and C . A and C are the arc ends, B is an inner point.

To draw the arc and to calculate the bounding box we need the center point M and the radius r .

The center point $M(x_M, y_M)$ is where the Mittelsenkrechten on \overline{BA} and \overline{BC} meet:

$$g_1: \quad \vec{r} = \vec{r}_B + \frac{1}{2}\vec{BA} + t\vec{r}_1 \quad t \in \mathbb{R}$$

$$g_2: \quad \vec{r} = \vec{r}_B + \frac{1}{2}\vec{BC} + s\vec{r}_2 \quad s \in \mathbb{R}$$

The Mittelsenkrechten are perpendicular to the vectors, so we have:

$$\begin{aligned} 0 &= \vec{r}_1 \cdot \vec{AB} & 0 &= \vec{r}_2 \cdot \vec{CB} \\ 0 &= \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \begin{pmatrix} x_B - x_A \\ y_B - y_A \end{pmatrix} & 0 &= \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \begin{pmatrix} x_B - x_C \\ y_B - y_C \end{pmatrix} \end{aligned}$$

This is fulfilled by:

$$\begin{aligned}x_1 &= y_B - y_A & x_2 &= y_B - y_C \\y_1 &= x_A - x_B & y_2 &= x_C - x_B\end{aligned}$$

We have the lines:

$$\begin{aligned}g_1 : \quad \vec{r} &= \frac{1}{2} \begin{pmatrix} x_A + x_B \\ y_A + y_B \end{pmatrix} + t \begin{pmatrix} y_B - y_A \\ x_A - x_B \end{pmatrix} \\g_2 : \quad \vec{r} &= \frac{1}{2} \begin{pmatrix} x_C + x_B \\ y_C + y_B \end{pmatrix} + s \begin{pmatrix} y_B - y_C \\ x_C - x_B \end{pmatrix}\end{aligned}$$

Where the lines meet in M we have:

$$\begin{aligned}x_B + \frac{1}{2}(x_A - x_B) + t_M(y_B - y_A) &= x_B + \frac{1}{2}(x_C - x_B) + s_M(y_B - y_C) \\y_B + \frac{1}{2}(y_A - y_B) + t_M(x_B - x_A) &= y_B + \frac{1}{2}(y_C - y_B) + s_M(x_B - x_C) \\ \frac{1}{2}(x_A + x_B) + t_M(y_B - y_A) &= \frac{1}{2}(x_C + x_B) + s_M(y_B - y_C) \\ \frac{1}{2}(y_A + y_B) + t_M(x_B - x_A) &= \frac{1}{2}(y_C + y_B) + s_M(x_B - x_C) \\ t_M(y_B - y_A) + s_M(y_C - y_B) &= \frac{1}{2}(x_C - x_A) \\ t_M(x_A - x_B) + s_M(x_B - x_C) &= \frac{1}{2}(y_C - y_A)\end{aligned}$$

This linear system of equations has either

- exactly one solution

$$t_M = \frac{\frac{1}{2} \left((x_B - x_C)(x_C - x_A) - (y_C - y_B)(y_C - y_A) \right)}{(y_B - y_A)(x_B - x_C) - (y_C - y_B)(x_A - x_B)}$$

or

- no solution (all three points on one line) or
- an infinite number of solutions (two or three of the points are equal).

2.1.4 Arrowheads

The need for a length correction

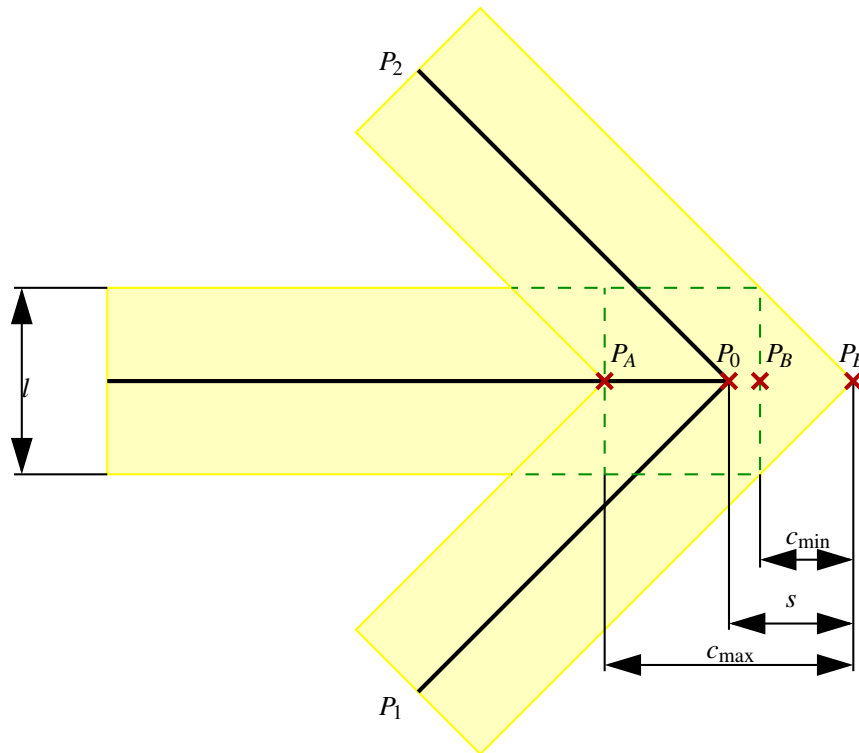


Figure 2.2: Length correction for arrowheads

For decorated lines the final (first) point in the Fig-file specifies the endpoint of the arrowhead. The line must be shortened a little bit before drawing it.

Figure 2.2 shows a line coming from the left and ending in P_0 . An arrowhead is attached to this line, drawing as a polyline $P_1 \dots P_0 \dots P_2$. To draw the arrowhead polyline we use the same line width as for the line itself. As a result of the drawing operation the arrowhead ends in point P_E .

If α is the angle of the arrowhead we can write

$$\frac{l}{2} = s \cdot \sin\left(\frac{\alpha}{2}\right)$$

$$s = \frac{\frac{l}{2}}{\sin\left(\frac{\alpha}{2}\right)}$$

For a given final point P_E , line width l and arrowhead angle α we can calculate the point P_0 using the distance

$$s = \frac{\frac{l}{2}}{\sin\left(\frac{\alpha}{2}\right)}$$

In the next step we can calculate P_1 and P_2 using the line width and height from the Fig-file.

Before we draw the arrowhead we need to draw the line itself. The line must end somewhere between P_A and P_B or on one of these points. The distance between P_A and P_E is $2s$, the distance c_{\min} between P_B and P_E can be calculated using:

$$\frac{\frac{l}{2}}{c_{\min}} = \tan\left(\frac{\alpha}{2}\right) \quad c_{\min} = \frac{\frac{l}{2}}{\tan\left(\frac{\alpha}{2}\right)} \quad c \leq s$$

$$c_{\max} = 2s$$

$$c = \begin{cases} s & \text{for the MetaPost driver using MetaPost arrowheads} \\ \frac{1}{2}(c_{\min} + c_{\max}) & \text{for all other situations} \end{cases}$$

Length correction on straight lines

To correct a line from $P_0(x_0, y_0)$ to $P_1(x_1, y_1)$ by c we can use

$$l = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \qquad q = \frac{l - c}{l}$$

for non-negative q we have:

$$x'_1 = x_0 + q(x_1 - x_0) \qquad y'_1 = y_0 + q(y_1 - y_0)$$

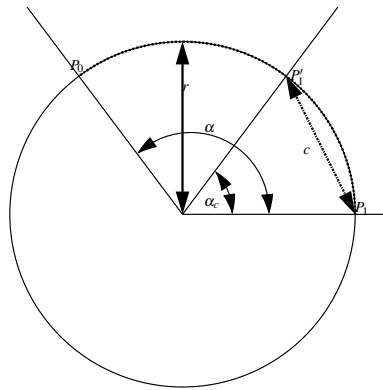
Length corrections on arcs

Figure 2.3: Corrections on arcs

For a given arc from P_0 to P_1 specified by radius r and angle α we can reduce the arc by decrementing the angle. P'_1 is the new final point of the arc, the required distance between P'_1 and P_1 is c .

The decrement value α_c can be calculated using

$$\begin{aligned}
 c &= \sqrt{(x'_1 - x_1)^2 + (y'_1 - y_1)^2} \\
 &= \sqrt{(r \cos \alpha_c - r)^2 + (0 - r \sin \alpha_c)^2} \\
 &= \sqrt{r^2 - 2r^2 \cos \alpha_c + r^2 \cos^2 \alpha_c + r^2 \sin^2 \alpha_c} \\
 &= r \sqrt{1 - 2 \cos \alpha_c + \cos^2 \alpha_c + \sin^2 \alpha_c} \\
 &= r \sqrt{2 - 2 \cos \alpha_c} \\
 \frac{c}{r} &= \sqrt{2 - 2 \cos \alpha_c} \\
 2 - 2 \cos \alpha_c &= \left(\frac{c}{r}\right)^2 \\
 2 \cos \alpha_c &= 2 - \left(\frac{c}{r}\right)^2 \\
 \cos \alpha_c &= 1 - \frac{c^2}{2r^2} \\
 \alpha_c &= \arccos\left(1 - \frac{c^2}{2r^2}\right)
 \end{aligned}$$

2.2 Notes

2.2.1 Font handling

The *handling* component in the *dk_fig_fonth_t* structure controls how to specify the font for a text.

The following values can occur:

- 0
 handling=none, font=tex
 Text is not processed by T_EX/L^AT_EX but a L^AT_EX font name is used.
 For the MetaPost driver this results in
 ...infont "ptmr" scaled ...
- 1
 handling=none, font=fig
 Text is not processed by T_EX/L^AT_EX, a PS font name is used.
 For the MetaPost driver this results in
 ...infont "TimesRoman" scaled ...
- 2
 handling=tex, font=tex
 Text is processed by T_EX/L^AT_EX, no font or text size is chosen explicitly.
 So the text appears in the L^AT_EX default font.
- 3
 handling=tex, font=fig
 Text is processed by T_EX/L^AT_EX, font and text size are used exactly as specified in the Fig file.
 \font\fntAA=ptmr at 12pt
- 4
 handling=tex, font=similar, size=fig
 Text is processed by T_EX/L^AT_EX, a font having similar features as the one specified in the Fig file is chosen automatically (roman/sans-serif/monospaced, normal/bold, upright/italic). The text size is set as specified in the Fig file.
- 5
 handling=tex, font=similar, size=tex

Text is processed by $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, a font having similar features as the one specified in the Fig file is chosen automatically (roman/sans-serif/monospaced, normal/bold, upright/italic). No size is specified, the same text size is used as in the enclosing document.