PostScript in groff

Introduction

Since computing's biblical times several flavours of the *roff* word-processor were the staple diet of slightly masochistic scientist, computer buffs and the like. These special folks did not mind to learn and experiment and getting frustrated at times, as long as they could then quickly produce documents that looked exactly how they wanted them to look, and earned them the awe of users of certain non-nix boxes.

From time immemorial they had macro packages for great looking complex math and tables, but were quite a bit short on matching graphics. This was a great handicap, especially for engineering buffs, who needed *eqn* and *tbl* too, but just could not properly explain their thoughts without complex sketches.

Enter Adobe and the *PostScript* printer. First some roff varieties started producing PostScript output, then buffs recognized that the graphics package they always wanted but their cruel bosses never bought for them was right under their noses, inside their PS printers.

Some of them took to hand-coding with delightful results and learned that PostScript's reputation of being a dog was thorougly undeserved. They learned that this reputation originated with *printer drivers* that produced literally hundreds of times bigger files than they could- and should have, and used techniques as if someone promised their creators iHuris for the worst result.

The pressure was on to give PostScript a place in the roff world, thus enabling our folks to Get Yeah Satisfaction. Here we discuss the coming together of *groff* and PostScript; a marriage made in GNU heaven. Getting through the pearly gates requires some learning and some pain in setting up the system but, if you are still here, you are *the type*, so what's the problem?

For those who are new to PS, the best starting point is the first edition of the PostScript Language Reference Manual. Amazon's partners still sell used copies for $2 apiece plus postage. Don't start with later editions, they are intimidating and are truly reference manuals, not teaching aids. Here we assume that you have some basic knowledge of PostScript and groff and that you have read the man pages of grops.

To PS or not to PS, what are the questions?

There are four main questions regarding the marriage of PS and groff:

- How to 'upload' PS code to various parts of the final PS version of your document, from within your groff code or from external PS files
- How to smuggle information known to groff into PS messages, e.g.
  via groff number registers
  via groff strings
  via groff parameters like those passed to groff macros
- How to pass groff's current position (CP) to the PS part and how to set it for the text that follows a PS 'insert'
- Apart from sharing space on paper, how to make your PS inserts and the groff/grops generated PS code independent of each other, that is a must with any substantial PS involvement

These questions and the corresponding answers form a web that is not that simple to linearize for easy learning. We'll try, but please be patient if something crops up that is not fully explained just now.

‘Where do we begin?’ sings Andy, and the Sisters answer him with the first principle of scientific thinking: ‘The facts, man, the facts.’ In attempting to untangle the web, it pays to walk this line.

What do we have: files ...

Ye Gods, we have them in spades. The fact that groff does not write PS code, but some intermediate thing, just adds to the variety. Thing is then converted to PS by *grops*, and grops needs some support too.

mydoc.grf	This is your groff source document that • may contain PS code, • may read-in PS code from external files, • may call macros that contain PS code, • may call macros that read-in PS code from external files.
mymacros	Your macro collection that you source-in at the beginning of your document, that • may contain PS code, • may read-in PS code from external files.
mypost	This is your ever-growing collection of general-use PS routines that you may use with your standalone PS programs or in conjunction with your groff documents. One needs to be careful not to let them interfere with grops’s own, e.g. by using dictionaries or save/restore pairs or by just using prefices in variable names.
prologue	This standalone file is a set of PS routines that grops prepends to its own PS code. • This is not the only PS code that grops prepends (see about this later). • It is good practice to create something like *prologue_my_own_copy* that you can freely modify (see about this later), and prepend with your latest PS routines, and • ask grops to use this instead of *prologue*, via the -P option.
DESC	The device DESCription file is home to a few very basic setup parameters, e.g. the default paper-size, and to an interesting thing, ‘broken’. The PostScript language is a set of instructions that describe a printed page. Every interpreter must know them well. However, these instructions do not impose a structure on the whole document, and that makes it difficult to know where exactly page 5 is, or what resources the program needs. Enter DSC (Document Structuring Conventions), that requires special comments in the PS program to state such things explicitely, at the very beginning. DSC has many advantages, e.g. the possibility of viewing PS documents out of sequence, and it is ‘the right thing to do’. If you have the time to do it, that is. Whilst groff complies (?), other applications may not. After all, it is not compulsory. The ‘broken’ thing adjusts grops’ output to accommodate a few known ‘offenders’. We linger here ’cause of potential problems. Best intentions are not always enough, the top of your pages may get clipped-off, etc. Then what do you do? We’ll see this later.
mydoc.ps	This is the PS program produced by groff-grops from your groff source and PS routines. • You may view this file with a PS viewer (e.g. *gv*), or send it directly to a PS printer. Use ‘lpr -l’ on the Mac to avoid unwanted ‘interpretation’. • You may convert it to a .pdf and print it on some new printers that can process such files natively (HP 4650, perhaps). • If you print the .pdf file on most other PS printers, first they will be converted back to huuuge PS files, and some uuughly yellowish background could be added to your carefully constructed scene. Arrrgh ...

Destination mydoc.ps

It is fitting to start with the structure of the end result, the PS file created by groff/groffs. Here we can pinpoint where exactly groff's different PS initiatives end up (e.g. 'ps: def ..' or 'ps: exec ..'), and discuss where you may wish to add your own PS routines and where you may wish to make some changes.

The anatomy of groff/grops' PostScript output

```
%!PS-Adobe-3.0                          A) This is the DSC header, created by grops at runtime
%%Creator: groff version 1.18.1
      :                                    Please note that though PostScript allows the mixing
%%Orientation: Portrait                    of portrait/landscape pages via the %%PageOrientation:
      :                                    comment, groff can only do either portrait or landscape
%%BeginProlog
```

```
%--- a few shorthands                    B) This is a good place for your own general use PS routines

/_b {bind def} bind def                    It is best to keep your routines in a separate file and
/_e {exch def} _b                          prepend it to grops' original prologue whenever you modify
      :                                    them or add to them, and ask groff to use this extended, and
/_u2mm {0.001 mul 2.8346 div} _b           perhaps modified, prologue
      :
```

```
/setpacking where { ..                  C) This is the prologue, a separate file of PS routines that
      :                                    grops copies here at runtime
/grops 120 dict dup begin                  You may wish to make some changes (see alternatives in blue)
      :
/BP {                                      Perhaps: /BP { <</PageSize [595 842]>> setpagedevice
/level0 save def                                    /level0 save def
      :                                                   :
/EBEGIN {moveto DEFS begin} def            Perhaps: /EBEGIN { save /pssave exch def moveto
                                                            matrix defaultmatrix setmatrix
                                                            72 25.4 div dup scale % hey, millimetres!
                                                            0.08 setlinewidth } bind def

/EEND /end load def                        Perhaps: /EEND   { pssave restore } bind def
      :
```

```
%%EndResource                           D) This is added to the prologue by grops on the fly
      :                                    We are still in the declarations phase
grops begin /DEFS 53 dict def
            DEFS begin                     Note: dictionary starts
/u {.001 mul} bind def
      :                                    Grops stores your \X'ps: def ..' definitions here
      :
end                                        Note: dictionary ends
      :
%%EndProlog
```

```
%%Page: 1 1                             E) Things start to happen here: this is the 'execution stream'
%%BeginPageSetup
BP                                         Begin page routine
      :
150 213 EBEGIN                             Grops passes current x y coordinates to EBEGIN and copies
      :                                    \X'ps: exec ..' instructions or \X'ps: file ..' files here
EEND                                       They get executed here then EEND closes your own PS session
      :
end
%%EOF
```

We shall return (to this list) a few times, but first: why change the prologue?

A few changes perhaps?

Page size. As things are (at least in version 1.18.1), you are expected to set the page-size in the DESC file, so that grops may translate it into something in the DSC header that the printer driver may understand and act upon.

However, there is no trace of page-size in this header (%%BoundigBox?, %%PageBoundingBox?), not even when DESC's 'broken' flag is set to zero. In the absence of such instructions you receive the default, that is likely to be US letter size.

If you live in an a4 country, the top of your pages will get clipped-off. Then you have one option left: to re-define the BP command by adding an explicit page-size via the *setpagedevice* PS operator as in blue in the previous chart. The [595 842] array shows the width and height of a4 in PS units (72 points/inch). Many interpreters/drivers have problems with recommended DSC comments, but no interpreter should have any problem with compulsory setpagedevice.

Protection. The other thing you may wish to change is the EBEGIN and EEND pair. They bracket all your PS code in the 'execution stream' of your final PS file. Currently they provide your PS inserts with a dictionary, that limits the scope of your '/variable something def' instructions, so that you may not spoil grops' own variables that may accidentally have the same names.

That's fine, but not enough protection. Your PS inserts may spoil the colour or font, they may leave inadvertently a path to be stroked later with unexpected result, or a clipping rectangle, etc. The proper separation of powers is via a save/restore pair that takes care of everything.

Millimetres. While at it, you may consider changing the scales to millimetres, perhaps you may feel much more comfortable with them. Remember that PS has real arithmetic, so there is no need to use huge integers à la groff here.

Problems? However, if you use save/restore to separate your PS from groff's, you will not have access to those definitions you put into Section D of the previous chart via \X'ps: def ..' instructions, [1] and you will not have access to the **u** conversion facility there either. Also, you will lose the ability to insert something PostScripty seamlessly into the text (e.g. funny characters). Whether all this constitutes a loss or not is much debatable and, if you wish to add some more serious graphics to your document, the save/restore pair is the way to go.

Since groff only uses integer arithmetic, it needs to scale-up numerical information before it stores them in number registers. The **u** process converts such upscaled numbers back to groff's point sizes. However, it is not very likely that you would continue to use groff's system in your PS inserts, so you will probably need to define your own scale-conversion routine. E.g. the **_u2mm** routine in Section B of the previous chart converts such upscaled numbers to millimetres.

Gee, how boring a politically-correct/scientific/sterile/naked page without a single PostScript highlight or insert can look like! Arrrgh!

[1] Had grops placed \X'ps: def ..' definitions outside of its own sphere of influence (e.g. in Section B of the previous chart), there would have been no such problem with save/restore.

XX and XY, what is going on here?

We have covered enough background to get closer to the details of the PS-groff marriage. We are interested in how to place information from groff into PS streams, how to upload these PS streams into mydoc.ps, and how to read-in external PS files. Here we find ourselves in X and Y territory.

One-liners. All the \X escapes are restricted to just one line. For example:

```
\X'ps: def    code'  \" upload 'code' into the definitions section, one-liner, 1 definition
\X'ps: mdef n code'  \" upload 'code' into the definitions section, one-liner, n definitions
\X'ps: exec   code'  \" upload 'code' into the execution stream   , one-liner
```

The stated *raison d'être* of the *mdef* escape is to tell grops how long a dictionary to create for multiple definitions. Yes, PS once insisted on properly sized dictionaries, not so since the second ice age.
To be redundant is not a problem, but since *def* and *mdef* are uploaded into grops' definitions section, you have to choose between them or a save/restore pair that properly separates your PS code from that of grops. And this is a problem.

Passing information from groff to PS. In all three cases above you may smuggle into the 'code' numerical or textual information from number registers and strings:

```
\n[num-reg]
\*[str-var]
```

The 'code' of *exec* gets executed immediately. Repeated execution with different smuggled-in info results in repeated, and different, actions.
The 'code' of *def* or *mdef* only gets stored. It is just sitting pretty waiting for a handsome 'exec' to Paso Doble with her. If you issue the same definition many times, with different smuggled-in info, then the last lovely will simply override all the previous ones.

Reading-in external PS files. Groff has two mechanisms to upload external PS files. In the two-step version you read the file into a macro/string first, then upload the content of the macro/string. This scheme allows for the passing of information from groff into the PS code using number registers and strings. To do this groff needs to observe backslashes, so this scheme is not good if you wish to read-in an ascii85 encoded image, that may contain lots of backslashes.

The second scheme is an \X escape that uploads the file immediately and vebatim. This can be used for the a.m. images but, of course, there is no point to put number register and string references in the file, no-one will replace them with values.

```
.   di  macro_to_receive_file        \" A1) Start a diversion for trf to read a file into a macro
.   trf ps_file_to_read_into_macro   \"     File may contain references to registers & strings
.   di                               \"     End diversion

\*[macro_to_receive_file]            \" A2) Upload the PS file with extra groff info embedded
-------------------------------------------------------------------------------------------------
\X'ps: file my_ps_file'              \" B)  Upload the PS file, immediately and verbatim
```

It should be noted that the 'file' escape's verbatim copying nature is not unconditional. If in the DESC file the 'broken' variable's value=2 bit is set, no lines of the included file beginning with %! will be uploaded. This may cause problems with ascii85 encoded images that may contain both characters and their combination. It would be nice to break this linkage between 'broken' and 'file' in future releases of groff.

More-liners. The \X escapes are fine, but being one-liners, they are of very limited use. For more substantial jobs groff provides the \Y escape that has a macro attached to it. This macro may have many lines of PS code in it. The general arrangement is like this:

```
.de macname                              \" start special macro
    ps: exec                             \" going into the exec stream
    < any number of lines of PS code >   \" may contain references to registers and strings
..                                       \" end special macro
```

```
\Y[macname]                              \" upload the special macro's content
```

A few things we should know about this construct:

- This macro must start with 'ps:', not even a blank line above it is allowed
- It may only contain PS code
- Though you may pass information to PS here via groff number registers and strings, this macro can not have parameters of its own à la ordinary macros

Embedding: hallelujah, brothers! The restrictions of no parameters and only PS content still make this construct of rather limited use. However, we may embed it in a 'proper' groff macro, that can have parameters, then the embedded structure too will have access to them. Here we can mix'n match PS and groff, can read-in files with/without number registers and strings, upload PS content, the works. Just we must help groff to find out where the embedded and embedding macros end. It's done with the help of a third, absolutely empty, macro. The general arrangement is like this:

```
.de end1                                 \" beg of auxiliary macro
..                                       \" end of auxiliary mac: there's nothing in it
```

```
.de embedding-normal-mac                 \" beg embedding macro, args e.g.: 15 "" "(text text)"
      < read PS files into macros: trf > \" file may contain \n[num-reg] \*[str-var] (single \)
      < find PS playground dimensions >  \" from height given as parameter
.de embedded-ps-mac end1                 \" beg embedded macro, ends at .end1  (not at ..)
      ps: exec                           %  no blank line above this
      < translate origin to lower/left > %  may contain \\$1 ... (embedding mac's args)
      < lotsa PS >                       %  may contain \\n[num-reg]
      < lotsa PS >                       %  may contain \\*[str-var]
      < \\*[mac-name-of-inc-file] >      %  may load macros containing external PS files
.   end1                                 \" end embedded macro
\Y[embedded-ps-mac]                      \" back in embedding macro, upload all this PS
      < \X'ps: file to-copy-verbatim >   \" e.g. for ascii85 encoded images, if any
      < calculate space for insert >     \" height of bottom line relative to page
      < get new page if necessary >      \" insert does not fit
      < set space for following text >   \" needs some experimentation (where is CP?)
..                                       \" end embedding macro
```

You may find that though this system works, it is far too large and far too complicated to be of use within your source document. If you want to keep your eye on the ball and don't want such structures overwhelm your source file, it seems a better idea to sacrifice self-containment, write a few general-use macros that can be spirited away into your library, and let these macros get their PS content from external PS files. Let's hope that sometime in the future there will be a groff-like creature designed with more natural- and wider use of graphics in mind, from the onset.

Current Position. Via EBEGIN groff passes the CP to the PS insert, so you can start PS-away relative to that. Yeah, but where is it? Relative to the text above the insert, relative to the bottom of the page (in case you need to know whether a new page is needed for the insert)? Things that come into the picture: whether you used a line break before getting into PS mode, whether \n[nl] contains the height of the bottom of the current line or the next line, whether \X or \Y escapes generated some extras for you.

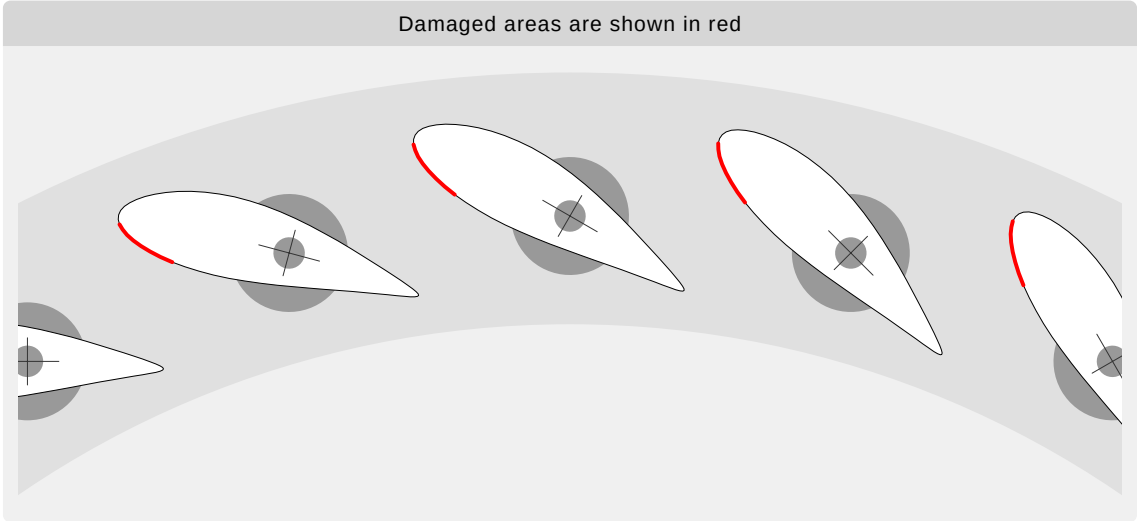
Sounds complicated and time consuming to follow. Fortunately Mr Heaviside taught us that we don't have to fully understand digestion, just to be able to eat. Usually you throw-away the x component of CP and, if there is a problem with the y component, you just add or subtract vertical spacing \n[.v] and, as we say it in Oz, she'll be all right, mate!

An annoying problem. There is an annoying problem with all the \X and \Y escapes, i.e. that they are escapes and, as such, they cause a mostly unwanted linefeed or an extra space in the text. E.g. if you write routines to plot a cross at the current position or at a point of given coordinates to check on the lay-out, this extra thing is a pain in the derrière. That's why Werner instituted the request equivalents of \X and \Y, '.device' and '.devicem', which will be available in the next release of groff.

Up to you. We have covered the essentials here but many details are still out in the cold. The problem is that things can be done zillions of ways. Instead of trying to cover a number of possibilities, we present just one characteristic example to give you the gist of things, and the rest is up to you.

Here's an example

Dear Mr Pie, On the 5th of June we inspected the guide vanes of your NF-2/A Francis turbine at Yarramunga Power Station. Extensive damage was found on all blades where indicated in the sketch below:



They look like cavitation but why-oh-why would cavitation damage be at such places? However, if some research grants were forthcomin', we would gladly run our world famous CFD programs to simulate the flow, cavitation

and erosion processes at different blade angles and operating conditions.

The job to do

Well, folks, the job is to develop a macro that paints the gray port for the guide vanes, but leaves the drawing of the blades to an external PS program-segment. The name of the PS segment will be passed to this macro as a parameter, so this macro can be re-used with other applications.

This little routine takes the Current Position from groff, places the origin for the PS program at the lower-left corner of the port, changes the scales to millimetres, calculates and applies the space that parts the before/after text. Quite a few sub-tasks to perform.

Of course, the PS segment too employs a few things that could be used in other applications, so these routines/definitions will be collected in a separate file to be prepended to the prologue once and for all.

The solution

Here are the listings of all relevant files, fully commented. These files are part of a tar package so that you may experiment with them and then develop your own solutions, according to your style and taste.

testme.grf: the source of the example

```
.
.so mymacs.lib  \" source my macs
.   po  3.2c    \" page offset
.   ll 15.0c    \" line length
.   ft TR       \" Times-Roman
.   ps 10       \" point size
.   pl 29.7c    \" page length (a4)
.   vs  0.5c    \" vertical spacing
.
Dear Mr Pie, On the 5th of June we inspected the guide vanes of your NF-2/A
Francis turbine at Yarramunga Power Station.
Extensive damage was found on all blades where indicated in the sketch below:
.
.ps_insert2 "(Damaged areas are shown in red)" 69 blades.inc
.
They look like cavitation but why-oh-why would cavitation damage be at such places?
However, if some research grants were forthcomin', we would gladly run our world
famous CFD programs to simulate the flow, cavitation and erosion
processes at different blade angles and operating conditions.
.   ex
```

mymacs.lib: the collection of a few macros

```
.\"-+----- reminder: layout registers
.\"
.\"        .p  page length                 .o   page offset
.\"        .v  vertical spacing            .ll  line length
.\"        nl  current vertical position
.\"
.\"-+----- an auxiliary
.
.de end1            \" end-of-macro macro
..
.\"-+----- define a few registers
.
.   nr _mm2u 2834   \" convert millimetres to "u" by multiplying with this
.   nr BM    3.0c   \" page: bottom margin
.   nr p_hh  0.6c   \" port: header height in mm
.   nr p_gt  0.5c   \"       gap at top (dist between text above and port)
.   nr p_gb  0.5c   \"       gap at bot
.
.\"-+----- read file into a macro/string (number registers and strings are ok)
.
.de file2macro1 \" calling: file2macro1 filename macroname
.
.\"                 \" dumping in ps: \*[macroname]
.\"                 \" dumping in gr:   .macroname
.   di  \\$2        \" beg diversion (let's assume that a .br is done)
'   trf \\$1        \" file may contain \n[num-reg] and \*[str-var]
.   di              \" end diversion
..
.\"-+----- current font: character heights in "u"
.
.de f_heights                       \" used in "after insert" spacing
.
.   nr tmp1  \\w'a'
.   nr f_ha  \\n[rst]               \" height of "a"
.   nr tmp1  \\w'H'
.   nr f_hH  \\n[rst]               \" height of "H"
.
.   nr f_hm (\\n[f_ha]+\\n[f_hH])/2  \" mean height, used in "after insert"
..
.\"-+----- calculate dimensions for ps ports
.
.de port_dim1   \" calling (after .br): port_dim1 port-height-mm-incl-header
.
.   nr p_yb \\n[nl]+\\n[p_gt]+(\\$1*\\n[_mm2u])   \" y-bot: y-cur + gap + height
.
.   nr p_pb \\n[.p]-\\n[BM]                       \" p-bot: page-len - bot-marg
.
.   f_heights                                   \" get character heights
.
.   nr p_dy \\$1*\\n[_mm2u]+\\n[p_gt]-\\n[.v]     \" transl ps to bot of port
.
.   nr p_sp \\n[p_dy]+\\n[p_gb]-\\n[.v]+\\n[f_hm] \" space to part text for port
.
.   if (\\n[p_yb]>\\n[p_pb]) .bp                 \" y-bot > page-bot -> .bp
..
.\"-+----- draw a gray port and upload a PS file to do some graphics in it
.
.de ps_insert2  \" calling: ps_insert2 "(header)" tot-height-mm inc-file
.
.   br                                          \" CP=(offs+ind,1-line-lower)
.   port_dim1   \\$2                            \" get dimensions
.   file2macro1 \\$3 inc_file                   \" PS file (regs & strings)
.
.de ps_insert2_aux end1                         \" PS macro within macro
    ps: exec                                    %  no blank line above this

%--- change x of CP to offset, origin at (offset, bottom-of-port)

    currentpoint exch pop \\n[.o] _u2mm exch \\n[p_dy] _u2mm sub translate

%--- (u1,v1) (u2,v2) corners of the playground, paint gray port, upload PS file

    /u1 0 def /u2 \\n[.ll] _u2mm def
    /v1 0 def /v2 \\$2 def

    u1 v1 u2 v2 \\$1 _port
    \\*[inc_file]
.end1
.
\Y[ps_insert2_aux]                              \" no leading space
.
.   sp \\n[p_sp]u                               \" make room for the port
..
```

mypost.lib: collection of a few PS routines

```
%--- a few shorthands

  /_b { bind def  } bind def
  /_e { exch def  } _b
  /_m { moveto    } _b
  /_d { lineto    } _b
  /_n { newpath   } _b
  /_c { closepath } _b

%--- convert upscaled groff units to millimetres
%
%    usage: something-in-u _u2mm -> something-in-mm

  /_u2mm { 0.001 mul 2.83464 div } _b

%--- draw horiz/vert lines
%
%    usage: x _h  or  y _v

  /_h { currentpoint exch pop _d } _b
  /_v { currentpoint pop exch _d } _b

%--- path of a curve stored in an array
%
%    usage: [x y x y ...] _r

  /_r { _n aload pop _m count 2 idiv { _d } repeat } _b

%--- store parameters of a process in the sequence they are given
%
%    usage: /a /b /c 3 defs to store the last 3 params in variables a, b, and c

  /_defs { 1 add -1 2 { -1 roll def } for } _b

%--- path of a rectangle
%
%    usage: x-left y-bot x-right y-top _rect

  /_rect { 4 dict begin /a1 /b1 /a2 /b2 4 _defs

           _n a1 b1 _m a2 _h b2 _v a1 _h _c end } _b

%--- draw a graphics/text "port": light gray box, darker gray header with title
%
%    usage: x-left y-bot x-right y-top (Title) _port

  /_port { 5 dict begin /a1 /b1 /a2 /b2 /tl 5 _defs /b2 b2 6 sub def

           a1 b1 a2 b2       _rect .94 setgray fill           % light big box
           a1 b2 a2 b2 6 add _rect .84 setgray fill 0 setgray % darker header

           /Times-Roman findfont 3.3 scalefont setfont        % font for title

           a1 a2 add 2 div b2 2.1 add _m tl stringwidth pop 2 div neg 0 rmoveto

           tl show                                            % title in the mid
  end } _b

%--- path of a full circle
%
%    usage: x-origin y-origin radius _circ

  /_circ { _n 0 360 arc } _b

%--- path of an annulus
%
%    usage: x-origin y-origin r-small r-big _annu

  /_annu { 4 dict begin /xc /yc /r1 /r2 4 _defs

            _n xc yc r1 0 360 arc xc yc r2 360 0 arcn _c end } _b

%---------
```

blades.inc: the PS insert that draws the blades

```
%-+------- geometry of blades
  /blade [              % coordinates: [x y x y ...]
  69.2  -3.8  68.8  -3.1  67.5  -2.3  65.3  -1.3  62.4  -0.2  58.6   1.2
  54.1   2.7  48.9   4.4  43.1   6.2  36.6   8.2  29.7  10.3  22.3  12.4
  14.6  14.3   6.6  16.2  -1.5  17.8  -9.8  19.1 -18.1  20.0 -26.2  20.4
 -34.2  20.4 -41.9  19.8 -49.3  18.8 -56.2  17.4 -62.7  15.6 -68.5  13.5
 -73.7  11.2 -78.2   8.8 -82.0   6.2 -84.9   3.7 -87.1   1.2 -88.4  -1.3
 -88.8  -3.7 -88.4  -6.0 -87.1  -8.3 -84.9 -10.4 -82.0 -12.5 -78.2 -14.4
 -73.7 -16.1 -68.5 -17.6 -62.7 -18.9 -56.2 -19.9 -49.3 -20.5 -41.9 -20.8
 -34.2 -20.8 -26.2 -20.4 -18.1 -19.6  -9.8 -18.6  -1.5 -17.4   6.6 -16.0
  14.6 -14.6  22.3 -13.2  29.7 -11.8  36.6 -10.5  43.1  -9.4  48.9  -8.4
  54.1  -7.5  58.6  -6.8  62.4  -6.1  65.3  -5.6  67.5  -5.0  68.8  -4.4] def
  /r1 497 def           % radius, annulus
  /r2 625 def           % radius, annulus
  /rb 552 def           % radius, blade placement
  /rh  30 def           % radius, blade hub
  /rs   8 def           % radius, spigot
  /al  30 def           % blade angle
  /nb  24 def           % number of blades
  /df 360 nb div def    % delta-phi: blade-pitch
%-+------- routine to draw a single blade
/draw-blade {                                          % draw a single blade
gsave df mul rotate 0 rb translate al neg rotate       % get it in position
  0.50 setlinewidth                                    % thin lines
  0.60 setgray 0 0 rh _circ fill                       % hub
  1.00 setgray blade _r _c gsave fill grestore         % blade inside
  0.00 setgray stroke                                  % blade contour
  0.60 setgray 0 0 rs _circ fill                       % spigot
  0.20 setgray -16 0 _m 16 0 _d 0 -12 _m 0 12 _d stroke % centre-lines
  1 0 0 setrgbcolor 2 setlinewidth                     % thick red lines
  blade 64 16 getinterval _r stroke                    % damaged area in red
grestore
} _b
%-+------- action: annulus + all blades (clipped)
gsave u1 2 add v1 u2 2 sub v2 _rect clip _n            % clipping rectangle
  u1 u2 add 2 div -103 translate                       % guide-vane sys origin
  0.26 dup scale                                       % scale the system
  0.88 setgray 0 0 r1 r2 _annu fill                    % draw annulus
  0 1 nb 1 sub { draw-blade } for                      % all blades, clipped
grestore
%---------
```

A setup file

```
#!/bin/csh
#
#                                        ****  *****  *****  *   *  ****     *
#                                       *      *        *    *   *  *   *   **
#                                        ***   ***      *    *   *  ****   * *
#                                           *  *        *    *   *  *        *
#                                       ****   *****    *     ***   *        *
#
#     Purpose..... Define aliases to process testme.grf with groff
#
#     Remarks..... You may need to modify this script depending on what
#                  system you may have
#                  You need to make this script executable and source it
#
#     Usage....... alias pro    will create an alternative prologue (admin/su)
#                  alias grf    will produce the PS output
#                  alias view   will show you the result
#
#-+------- please edit these three lines

  setenv dir_try /Users/ms1/wrk/grf/tar                 # name of cur directory
  setenv dir_pro /usr/share/groff/1.18.1/font/devps     # home of prologue
  setenv dir_tmp /Users/ms1/tmp                         # dir of temporaries

#-+------- designate a file for the PS output

  setenv output $dir_tmp/testme.ps                      # output file

#-+------- concat mypost.lib & prologue.mod and place it where grops expects it
#          you may need to be admin or su to source this file & use this alias
#          you may need to change owner/group of the the file

  alias pro  'cat $dir_try/mypost.lib'                  # alternative prologue\
                '$dir_try/prologue.mod > $dir_pro/prologue.mod'

#-+------- typing "grf" will produce the PS output

  alias grf  'groff -s -I $dir_try -P -Pprologue.mod'   # produce PS output\
             '$dir_try/testme.grf > $output'

#-+------- viewing the output file with the Mac's Preview

  alias view 'open -a Preview $output'                  # if you have Preview

#---------

  exit

#-------------------------------------------------------------------------------
```

Epilogue

Finally a thank-you is due to many. Transitioning to groff only lately I had lots of questions regarding the conversion of my old troff-PS things. Werner and many other forum members were very helpful and very patient. This documentation is based on what I learned from them in little chunks in the last few months. Thank you all.

Miklós Somogyi