PROGS
Professional & Graphical Software

Tel/Fax (016) 48 89 52
Haachtstraat 92
3020 Veltem
Belgium

PROforma

PROGS Font & Raster Manager

written by :
Joachim & Nathan Van der Auwera

2 February 1994

Contents

1. Introduction 1
  1.1 What is PROforma ? 1
  1.2 Disclaimer & Copyrights 1
  1.3 Present, Past and Future 2
  1.4 This manual 3
  1.5 Installation 3
2. PROforma 7
  2.1 Concepts 7
    2.1.1 Graphics State - Gstate 7
    2.1.2 Driver & Device 7
    2.1.3 Path 8
    2.1.4 Subpath 8
    2.1.5 Path segment 8
    2.1.6 Bezier curve 8
    2.1.7 Clipping path 9
    2.1.8 Transformation matrix 9
    2.1.9 User space 11
    2.1.10 Device space 11
    2.1.11 Current point 11
    2.1.12 PageBbox & PageOrigin 11
    2.1.13 Font. 12
    2.1.14 Fontmap 12
    2.1.15 Fontlist 12
    2.1.16 Font Caching 13
    2.1.17 Extended Character Set 13
    2.1.18 Kerning. 13
    2.1.19 Ligatures 14
    2.1.20 Tracking 14
  2.2 Imaging Model 14
    2.2.1 Pages 14
    2.2.2 Pixels 15
    2.2.3 CRT screen 16
    2.2.4 dot matrix printer 16

2.2.5 inkjet or bubblejet printer. 17
2.2.6 laser printer. 17
3. Graphics 19
3.1 Gstate 19
3.2 Drivers 20
3.3 Transformation matrix 20
3.4 Drawing parameters 22
3.5 Clipping path 24
3.6 Building a path 24
3.7 Controlling the visible area 25
3.8 Controlling the page 27
3.9 Displaying pictures 28
4. Font Management 29
4.1 Font Loading 29
4.2 Font Information 30
4.3 Available Fonts 31
4.4 Text display 31
4.5 Cache handling 32
4.6 Charpaths 33
5. PROforma sessions 35
6. Operators 39
6.1 Parameters 39
6.1.1 Strings 39
6.1.2 Fixpoint - pt 40
6.2 List of Operators 40
6.2.1 PFAllFontCount 40
6.2.2 PFCharAvailable 40
6.2.3 PFCharPathEl 41
6.2.4 PFClearClip 41
6.2.5 PFClearPage 41
6.2.6 PFClosePath 41
6.2.7 PFCurveTo 42
6.2.8 PFDeviceColour 42
6.2.9 PFDriverCount. 42
6.2.10 PFFlatness. 42
6.2.11 PFFontBbox 42
6.2.12 PFFontCount 43
6.2.13 PFGetFontFamily. 43
6.2.14 PFGetFontName 43
6.2.15 PFGetFontNotice 43
6.2.16 PFGetFontVersion 43
6.2.17 PFGetFontWeight 44
6.2.18 PFGetPageBbox 44
6.2.19 PFGrayShade 44
6.2.20 PFInitCharPath 44
6.2.21 PFJustKernShow 44

6.2.22 PFJustShow 45
6.2.23 PFKernShow 45
6.2.24 PFKernWidth 45
6.2.25 PFLineTo 45
6.2.26 PFLineWidth 46
6.2.27 PFLoadFont 46
6.2.28 PFMoveTo 46
6.2.29 PFNextDriver 46
6.2.30 PFNextFont 46
6.2.31 PFNextFontName 47
6.2.32 PFNoCache 47
6.2.33 PFNrOfCopies 47
6.2.34 PFPathClear 47
6.2.35 PFPathClip 47
6.2.36 PFPathDraw 48
6.2.37 PFPathEOFilled 48
6.2.38 PFPathFast 48
6.2.39 PFPathFilled 48
6.2.40 PFPathStroked 48
6.2.41 PFRcurve 48
6.2.42 PFRemoveGstate 49
6.2.43 PFResetCTM 49
6.2.44 PFResetPageBbox 49
6.2.45 PFRestoreCTM 49
6.2.46 PFRestorePageBbox 49
6.2.47 PFRline 50
6.2.48 PFRmove 50
6.2.49 PFSaveCTM 50
6.2.50 PFScaleFont 50
6.2.51 PFScalePage 50
6.2.52 PFScrollPage 50
6.2.53 PFSelectFont 51
6.2.54 PFSetCTM 51
6.2.55 PFSetPageBbox 51
6.2.56 PFSetPageOrigin 51
6.2.57 PFShow 52
6.2.58 PFShowPage 52
6.2.59 PFShowPicture 52
6.2.60 PFTrackShow 52
6.2.61 PFUnLoadFont 53
6.2.62 PFUseCache 53
6.2.63 PFWidth 53
6.2.64 PFWindowMove 53
6.2.65 PFXShow 53
6.2.66 PFXYShow 54
7. Examples 55

7.1 test 55
7.2 cachetest 55
7.3 fonttest 56
7.4 pagetest 56
7.5 showfont & showfonts 57
7.6 chartest 57
7.7 charpathtest 57
7.8 picttest 57
7.9 cliptest 58
7.10 unicode_c 58

of making a backup.
Although much care is taken in the development of the PROforma software and manual, in no circumstances will PROGS, Professional & Graphical Software, be liable for any direct, indirect or consequential damage or loss arising out of the use or inability to use the PROforma software or documentation.
This said, it speaks for itself that PROGS will continue to develop this manual and software. Therefore, we would appreciate any comments about our software and manual. As you may know, we are only human, we can do no more than our best to provide you with the best quality software. If we do notice some inconsistencies between the documentation and the software, there may be an additional file on the program disk (updates_doc and/or updates_txt).

1.3 Present, Past and Future

PROforma is originally developed as the graphics library for LINEdesign. When we started developing LINEdesign v2, we felt that the graphics routines we used were too slow, and also too restrictive. On the other hand, LINEdesign v1 was quite greedy on memory. Therefore, we threw away all the old routines, and started writing a new, more powerful and faster set of routines. During this development, we even introduced some concepts (like the clipping path), which are not used in LINEdesign. On the other hand, the graphics library was expanded to allow efficient editing on screen.

So what do we have now ??

We have a system that can efficiently render and display fonts. All fonts can be shared among applications. A font cache is used to speed up the handling of fonts. Even the font cache and everything in it is shared amongst applications.
The system can draw lines and curves either stroked (with given accuracy and thickness), or filled (using either in/out or winding rule).
Anything can be displayed in any gray shade. If wanted, everything can also be clipped by regular or irregular shapes. Transformation matrices can be applied on the page.
The user can define which part of the coordinate space is actually visible on page (or screen).
Bitmaps can be directly displayed. This allows the user to include screens in his or her output.

What do we want for the future ??

Although we have done our best to provide the best quality and speed possible, we do hope we can improve the speed even further. On the other hand, we feel a need to improve the quality of fonts, especially at small sizes. We hope we will be able to do this by improving the hinting mechanism and hints in the fonts.
The current system is completely black and white. We plan to make it full colour, allowing

want you can specify the device which should be used to search for the PFfontmap file. So the command :

EX flp1_PROforma;'flp2_pf_'

will load PROforma from 'flp1_' and load the PFfontmap file as 'flp2_pf_PFfontmap'. If you want, the ending underscore can be left out.

On the disk there is also a PFConfig program. This is used to configure PROforma (meaning, to change the 'PFfontmap' file). This program loads the PFfontmap file when loaded (just like PROforma does, and saves it back, containing the new information, when it finishes. The operation is quite straightforward. This program allows you to :

• set the search path for fonts. Fonts which have to be loaded by PROforma are searched for on the given devices. You can specify more than one device if you separate the devices by semi-colons (';'). Note that the devices have to include the ending underscore ! The listed devices are searched from left to right. This may be important if there is a file with the same name in more than one directory.

• You can set the (maximum) amount of memory which may be used by a buffer. For instance in LINEdesign v1 128k was allocated for the printer buffer. However, in PROforma, this is equivalent to a 64k buffer (because of more efficient memory usage). On an average 300dpi printer this will result in 17 passes. However, as another difference with LINEdesign v1, you can configure the amount of memory needed, and no more will be used, usually even less, as only the actually used amount is allocated. If you want to use this option, you should make sure 'maximum' is also indicated.
However PROforma can also use as much free memory as it can get, while still keeping some free for fonts, other jobs, and some memory needed for rendering complex images (or fonts). If you want to make sure that a minimum amount of memory is left free, then you also have to specify the 'minimum' option.
For instance 'minimum' and 64 would always leave 64k free when allocating a buffer. If you would have specified 'maximum' and 128, then at most 128k would be used for the buffer. As can be seen from this example, all sizes are given in kilobytes. We do advice always to use figures above twenty.

• Cache size can also be configured. To speed up the display of fonts a special mechanism called a font cache is used. This means that (within certain restrictions) when a character is displayed, it is also kept in the font cache in a special form so that it can be reproduced quickly at the same size and rotation. However, this can take quite a lot of memory. For this reason the font cache can be shared between applications. You can give any size to the font cache. We advice the font cache to be at least 64k long, however if you can spare the memory you could make the font cache larger. If you can not spare any memory for the font cache you could specify a zero font cache size.

• Add font to fontmap. This command allows you to make sure a certain font can be used by PROforma. When you indicate this command, you get a directory in which you can indicate a PROforma font file. This font will then be added to the fontmap. Please note that the device where the font can be found should also be included in the search path or

On devices : we strongly recommend the use of the parallel printer port and not the serial port. Serial ports are extremely slow and the amount of data which has to be sent to a printer can be huge. Of course we try to send as little data as possible, but not too many printers can handle compressed data. You should also be aware that serial to parallel converters do NOT speed the transfer of data up. The serial port can handle a certain speed and not more. For instance try sending an A4 page of 300 dpi data on a 9600 baud serial port (standard). This A4 page would need about 966k of data and this would take at least 13 minutes without control bits or correction of control bits (and without handshaking). In short, it will take MORE than 13 minutes to send this data. Luckily, PROforma will normally send less than 966k.

2.1.3 Path

There are actually two meanings for this term, a device interpretation and a graphical interpretation.

- device : a device name, possibly including directory, where a file(s) can be found. In PROforma we also allow semicolons in a path name to distinguish between several paths to form a searchpath (that is all paths are tested from left to right until the requested file is found).
- graphical : a collection of subpaths.

2.1.4 Subpath

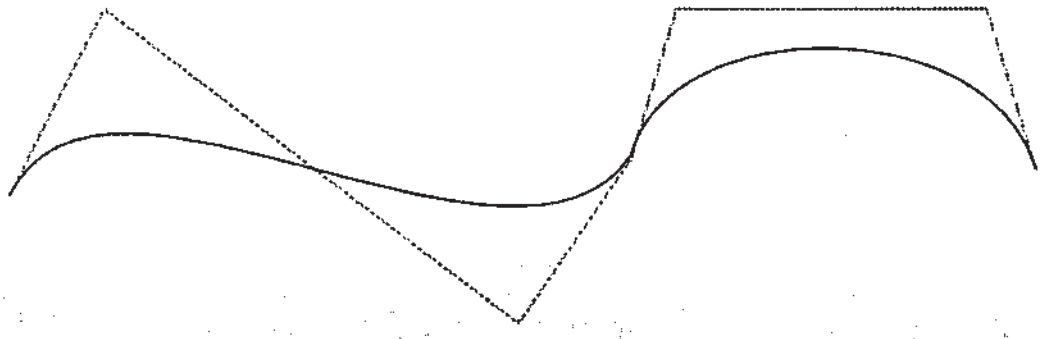
A move (to define the origin of the subpath) followed by a sequence of path segments. A subpath can be open, or closed.

2.1.5 Path segment

A path segment is either a line or a bezier curve. Circular arcs are converted to bezier path segments.

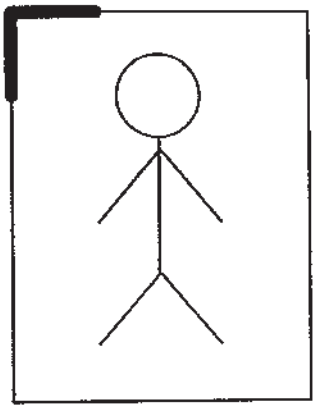
2.1.6 Bezier curve

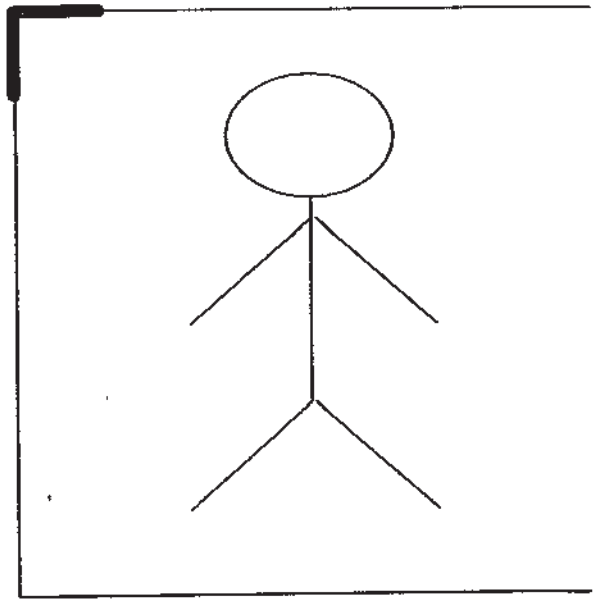
Bézier is a French mathematician who works for Renault and who "invented" a description/display method for curves based on Bernstein polynomials.
In PROforma we only use cubic bezier curves. That is curves which consist of four points: the two endpoints (which are on the curve), and two controlpoints (which are off the curve).



$$\begin{pmatrix} \text{x-scale} & 0 \\ 0 & \text{y-scale} \\ 0 & 0 \end{pmatrix}$$

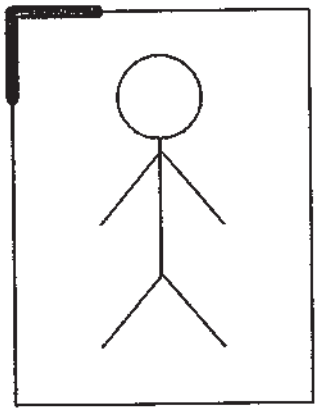
x-scale = 2
y-scale = 1.5

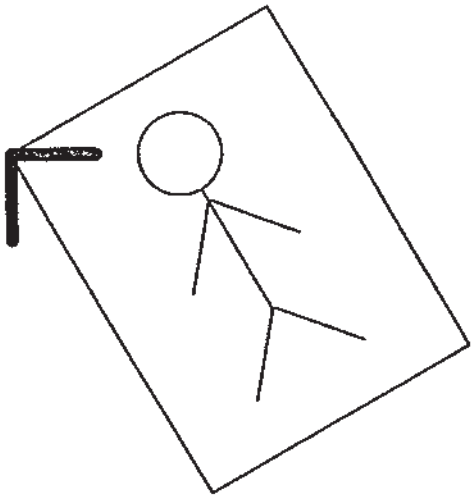




$$\begin{pmatrix} \cos\omega & -\sin\omega \\ \sin\omega & \cos\omega \\ 0 & 0 \end{pmatrix}$$

$\omega = 30°$



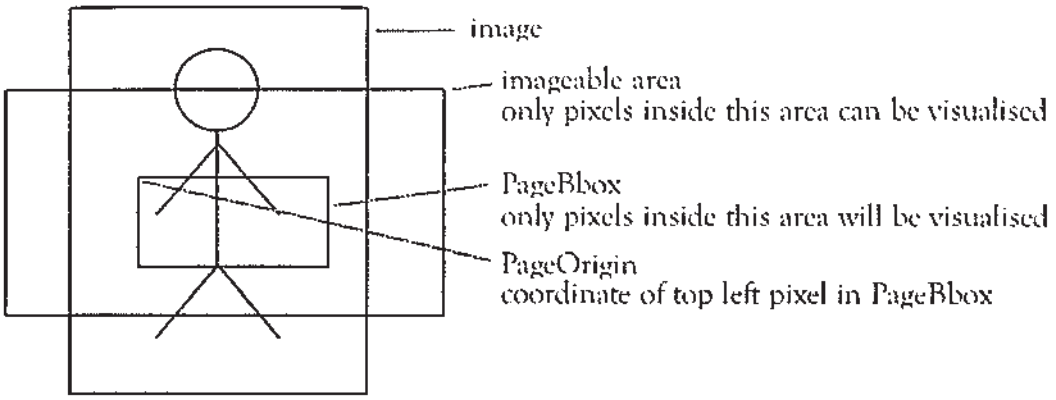


To combine : multiply from left to right
put the first transformation on the left,...

$$\begin{pmatrix} xs1 & yr1 \\ xr1 & ys1 \\ xd1 & yd1 \end{pmatrix} \times \begin{pmatrix} xs2 & yr2 \\ xr2 & ys2 \\ xd2 & yd2 \end{pmatrix} = \begin{pmatrix} xs1\times xs2+yr1\times xr2 & xs1\times yr2+yr1\times ys2 \\ xr1\times xs2+ys1\times xr2 & xr1\times yr2+ys1\times ys2 \\ xd1\times xs2+yd1\times xr2+xd2 & xd1\times yr2+yd1\times ys2+yd2 \end{pmatrix}$$

To convert back to default coordinates :

$$(x\ y) \times \begin{pmatrix} xs & yr \\ xr & ys \\ xd & yd \end{pmatrix} = (x\times xs+y\times xr+xs\ \ x\times yr+y\times ys+yd)$$



2.1.13 Font

Collection of graphical shapes, which can usually be combined to give readable text. The font files currently have a lot of similarity with the Adobe Type 1 font format (slightly adopted for easier access, which also makes them a bit shorter). However this may change in future if we choose to add a different hinting scheme (as the hinting used in type 1 files is quite obscure, and our current implementation quite unsatisfactory).
Fonts are handled quite efficiently. Each font will only be in memory once. Clients have to state which font they want to use (load), or no longer want to use (unload). Fonts are always referenced by their name. The name of the font and where to find it are stored in the "fontmap." The fontmap is read when PROforma is loaded. If a font is not in the fontmap, then it can't be used.
PROforma automatically releases a font when there are no gstates which have loaded it. Special routines are included to make sure this is always true (even when a job is force removed). When a font is loaded it is placed in the "fontlist" for that gstate.

2.1.14 Fontmap

PROforma always keeps a table of all known fonts. This table is used to map a fontname to a fontfile. If a client tries to access a font which is not in the fontmap, then an error is returned.
The fontmap can not change after PROforma has been loaded (except by removing the PROforma job and loading it again, alas this also removes all clients of PROforma).
Naturally, the fontmap can be examined to find out which fonts can be loaded (if the fontfile is available or fonts is already loaded of course).

2.1.15 Fontlist

Each gstate also keeps a list of the fonts which it can already access. A gstate can only access fonts which are actually loaded. Therefore, when the client request to load a font, it is added to the fontlist of the gstate. The fontlist can be examined to find out which fonts can already be used by a gstate.

2.1.19 Ligatures

Another typesetting feature is that some characters sequences like "ff", "fi", "fl", "ffi", "ffl" should be replaced by special characters which look better. Ligatures are supported in the Extended Character Set and can therefore be used by the client.

2.1.20 Tracking

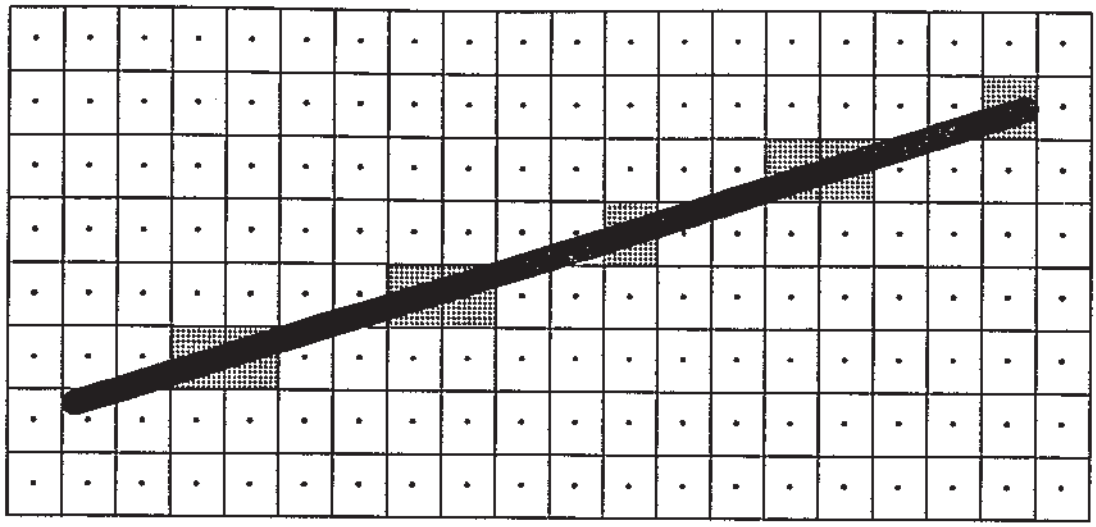
Sometimes it may be interesting to add some extra space between all characters. This is called tracking, and can be particularly useful for logo's.

2.2 Imaging Model

PROforma has it's own specific way look at pixels and pages, the two basic entities in this system.

2.2.1 Pages

Because not all devices are capable of changing their output (printers for example), PROforma uses a buffered approach. So instead of drawing on a page, all operators actually draw in a buffer, and this buffer can then be displayed on the actual page (PFShowPage). However, such a buffer can be quite large (typically 1MB for a 300dpi A4 page), and there may not be enough memory available for the entire page. Therefore, an actual page can be split into several pieces, and transferring the buffer will only display part of the page.
So pages are built in passes. The client knows how many passes are necessary for each page and has to call the display operators for all visible objects on the page once for each pass.



In the picture you see a line which is less than a pixel wide (and not hairline), and which pixels would be drawn.

The same rules apply to stroked paths. However when the linewidth is less than one pixel, the path will be drawn hairline. A hairline is a line with a uniform width of one pixel.

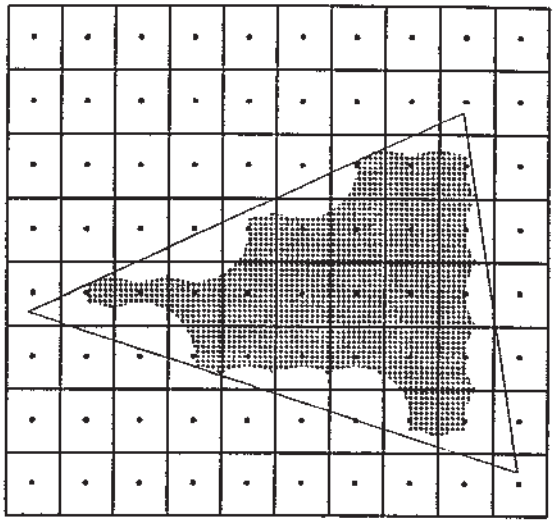
Unfortunately, the view that PROforma has on pixels is ideal and does not conform with most output devices (actually, I think only LCD screens work like this). There are two differences possible.
For starters, some devices don't draw their pixels as PROforma does it, but at the actual crossing of the grid lines. This is no problem as it only means there is a shift of half a pixel for the entire page. This causes no problems at all.
On the other hand, pixels are usually round, and they often overlap. To make matters even worse, some printers don't even have a consistent pixel size. We will just explain what the problems are with a few types of devices.

2.2.3 CRT screen

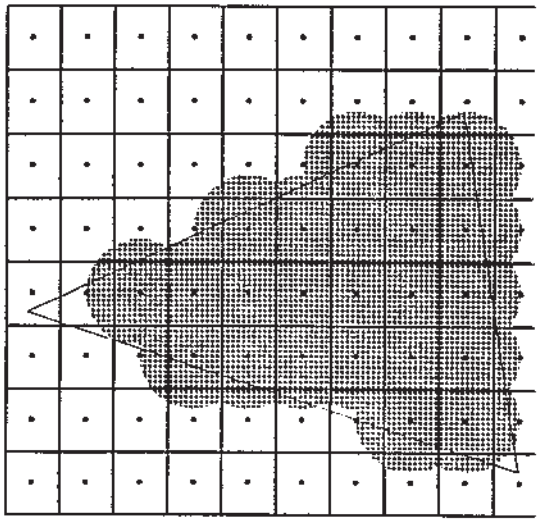
These are the common monitors, and we are lucky. Monitors draw in white, which has the effect that white pixels are larger than black pixels. However, the difference in size is not too large. The average size of the dots is slightly bigger than the addressable resolution. This is quite a good approximation of the PROforma model.

2.2.4 dot matrix printer

Dot matrix printer have round dots which are always equal in size. Dots are usually much larger than the resolution at which they are positioned. Although this produces smoother results, it also meant than output is usually more black than is intended. For instance the difference between a one or a two pixel wide line can be very small, even if this is a relatively big difference in user coordinates. See picture :



white draw laser printer



black draw laser printer

Most internal structures are initialised properly by PROforma like the flatness, device colour (black), gray shade (black),... However some things are not initialised like the linewidth, and the path type (so this should be set or a path will not become visible).

PFRemoveGstate

Remove the given gstate from memory. This releases all the fonts which are not used by other gstates, and also releases the current path,...

3.2 Drivers

PROforma also includes routines to enquire about the available drivers. This can be necessary because driverid's are not fixed. They can very between versions and/or configurations of PROforma. The only fixed driverid is currently zero (screen driver). A disadvantage of this approach is that you have to create a gstate before you can enquire the device. This may in future be solved by introducing a dummy device. On the other hand the client can always open a screen gstate.

PFDriverCount

Get the number of available drivers. This count does not include the screen drivers as this has a fixed driverid (zero). This command also assures that the next call to PFNextDriver for this gstate will return the first driver in the list.

PFNextDriver

Get the driverid and name of the next driver in the list. This list is not sorted !

3.3 Transformation matrix

To make it easy for a client to produce moved, slanted, scaled, rotated,... images, PROforma uses a transformation matrix. This matrix converts given coordinates to coordinates in default user space (which are then, internally, converted to device space). There is always a current transformation matrix. The default does nothing (unity matrix).
To allow the client to set the matrix to a certain state, which can be altered and later recovered, it is possible to save and restore the CTM.

PFMoveCTM

Move the origin of the current transformation matrix. This means that all objects are actually moved over the given distance. This command could be simulated by adding the given coordinates to all following absolute coordinates. This routine is a macro for PFSetCTM.

3.4 Drawing parameters

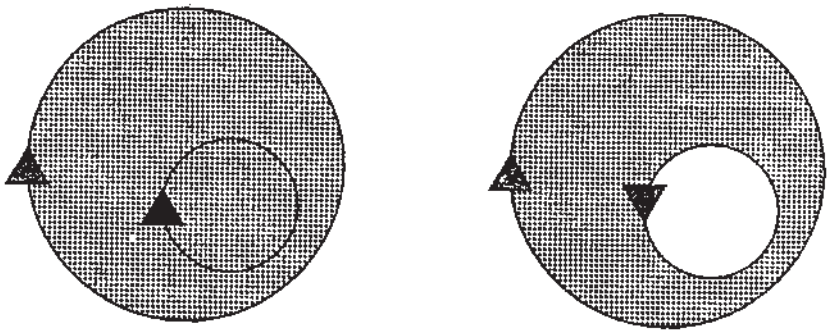
Because PROforma actually attempts to produce pages as fast and efficient as possible, the client has to know how the path has to be visualised before it is actually built.

PFPathStroked

Notify PROforma that all future path construction commands will apply to stroked paths. Stroked paths have a thickness (as specified by PFLineWidth), and always have round caps and round joins. Stroked lines are always visible. This means that even zero width lines are drawn one pixel wide (also called hairline). This is done to make sure that they don't disappear from the final result. Note however that hairlines can be so thin on certain high resolution devices that they may be invisible. Also some devices (like laser printers) are very bad at colouring small areas, which can have the same result.

PFPathFilled

Notify PROforma that all future path construction commands will apply to paths which are filled using the winding rule. Please note that areas which are less than a pixel wide or tall can disappear from the final result.



To determine whether an area is filled by the winding rule. Initialise the winding counter to zero and draw a line to infinity. For every edge which is crossed you should add one to the winding counter if the edge goes up (left if the edge is horizontal). If the edge goes down (right for horizontal), then you should subtract one from the counter. The area will be filled when the winding counter is not zero. All areas have to be closed for this rule to work. The direction of the path is very important as can be seen in the picture.

PFPathEOFilled

Notify PROforma that all future path construction commands will apply to paths which are filled using the even odd rule. Please note that areas which are less than a pixel wide or tall can disappear from the final result, see "PROforma Imaging Model."

3.5 Clipping path

To aid in special constructions, you can define a clipping path. This means that of any following drawings, only the parts which fall inside the clipping path are visible.

PFClearClip

Clear the clipping path and the current path (initial state). The parts of the path which are not inside the PageBbox will never be visualised, irrespective of the clipping path.

PFPathClip

Convert the path which was built into the current clipping path. The path will actually be clipped according to the previous clipping path. The path will also be cleared by this command, and the current point will be reset.

The path will be clipped as drawn with PFPathFilled unless the current drawing mode is PFPathEOFilled. If PFPathFast is selected or the linewidth is hairline (less than a pixel thick), then the clipping path can make everything invisible!

Clipping paths are a powerful tool. They allow you to look through an area. It can be viewed as if the drawing which you draw afterwards is the building of a large pattern which is used to fill the area indicated by the path which is clipped.

3.6 Building a path

Of course you also need commands to build a path. This path can then be drawn, or used as a clipping path. It is strongly advised that no other operators are called during the building a path, and between the building of the path and the actual drawing or clipping. If you do call other operators, the behaviour may be quite unexpected, see "PROforma sessions."

PFMoveTo

Set the current point to the given absolute position. If you were drawing a filled path which was not closed, this will be done automatically. This commands actually starts a new subpath.

PFRmove

Move the current point by the given distance. If you were drawing a filled path which was not closed, this will be done automatically. This commands actually starts a new subpath.

PFLineTo

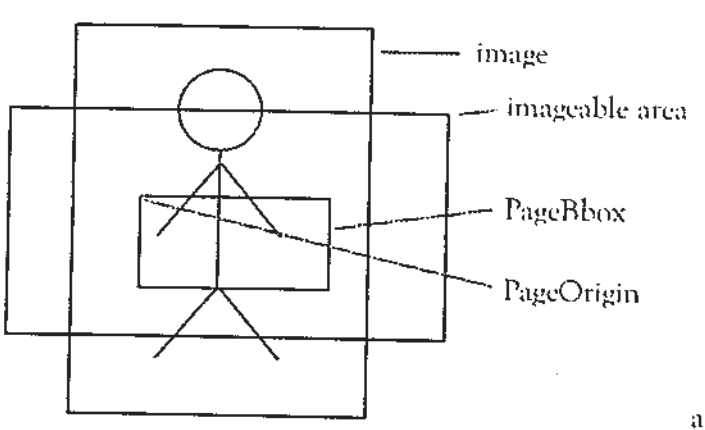
Construct a line from the current point to the given point in absolute coordinates. After this command, the endpoint will be the new current point.

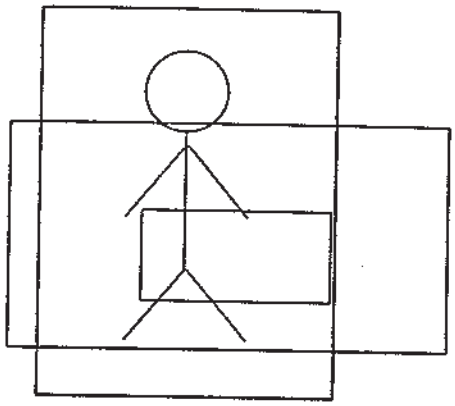
PFSetPageBbox

Set the new size of the PageBbox and moves the origin (relative to previous value) of visible area. All parameters to this function are in the default user space.

PFSetPageOrigin

Move the PageOrigin relative to the previous value. The PageOrigin is the coordinate of the top left point in the pagebbox. All parameters to this function are in the default user space.



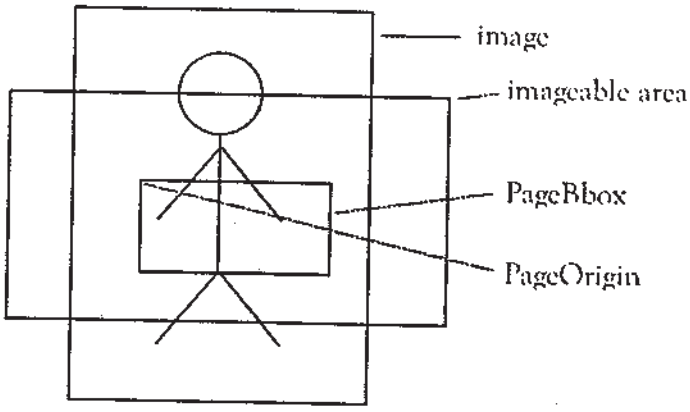


after PFSetPageOrigin with positive parameter

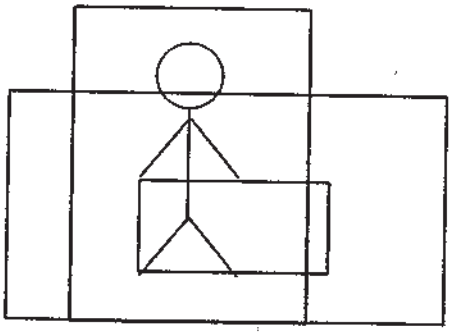
PFScalePage

This routine allows the client to set the scaling of a page. It is useful in cases where the page is zoomed in (as in LINEdesign). The default user space is actually changed from the previous value (initially 72 points/inch) to something else (e.g. to 144 pt/inch if factor was 2). Please note that this changes both the size of the PageBbox and the PageOrigin. Both are multiplied by the inverse of the scale.

PFScalePage allows the client to transparently change the imaginary size of the page. Contrary to PFScaleCTM which has no effect on any parameters in default user space (such as for PFSetPageBbox).



after PFScalePage, factor .8



PFGetPageBbox

Get the current size and origin of the Pagebbox, and the current value of the PageOrigin. All returned

3.9 Displaying pictures

A special library call is provided to display bitmaps. This is particularly important for DTP applications. Bitmaps can be visualised in any orientation.

PFShowPicture

Used to let PROforma display a bitmap picture (in PostScript terms this is a sampled image). There are several types of bitmap pictures in the computer world and some of them are directly supported. The picture will be displayed at the current position, using the CTM, at the requested size. If the client wishes to do so, an array can be specified to choose which "colours" have to be mapped to which grayshades.

Select the font with given name as current font. The font will be scaled to 1 point.

PFScaleFont

Scale the current font to the given point size. Scaling a font is always relative with the CTM.

4.2 Font Information

PFGetFontName

Get name of current font

PFGetFontFamily

Get name of font family of current font

PFGetFontVersion

Get version of current font

PFGetFontWeight

Get weight of current font

PFGetFontNotice

Get notice of current font. This usually includes details about creator and/or copyright on the font.

PFCharAvailable

Routine to allow the client to know whether a character (with given unicode) is available in the current font.

PFWidth

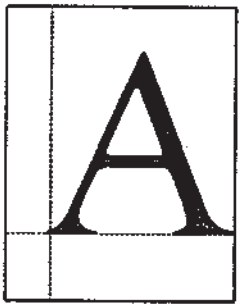
Get the width of a string.

PFKernWidth

Get the width of a string, when displayed with kerning.

PFFontBbox

Get the FontBbox. This allows you to find out the maximum amount a character can extend to right and top, and to left and bottom. The results for this command have an inverted y-axis.



FontBbox (black)
and offset to character origin (gray)

PFXShow

Show the requested string at the current position. The advance width of the characters is overwritten with the given values. The vertical advance width is taken as zero. The current position will now be the place where the next character should be placed.

PFXYShow

Show the requested string at the current position. The advance width of the characters is overwritten with the given values. The current position will now be the place where the next character should be placed.

PFTrackShow

Show the requested string at the current position. The current font size is used, but characters are positioned in such a way to make sure that the string has the given width.

PFJustShow

Same as PFShow, except that some whitespace is added or subtracted at the space characters, to make sure that the text is displayed with full justification. The requested width of the line has to be specified.

PFJustKernShow

Same as PFKernShow, except that some whitespace is added or subtracted at the space characters, to make sure that the text is displayed with full justification. The requested width of the line has to be specified.

4.5 Cache handling

Normally all font display operators try to use the font cache. However, as there are some problems with the usage of the font cache, the use of the font cache can also be switched off. This can be done because all font display operators work as if there is no clipping path when the cache is switched on !

PFUseCache

Instruct PROforma to try to use the font cache. This means that the font operators will not be clipped.

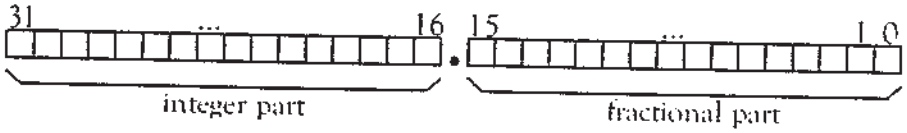
PFNoCache

Don't use the font cache, even when it's possible. This allows fonts to be clipped.

```
        < Draw Page >
    }
    PFRemoveGstate

< Draw Page > =
    /* FOR each pass */
    {
        [ PFClearPage ]
        < Draw Pass >
        PFShowPage
    }

< Draw Pass > =
    /* FOR each object */
    {
        < Draw Object >
    }

< Draw Object > =
    [ PFScalePage ]
    [ PFSetPageBbox | PFSetPageOrigin ]
    [ < Set Draw Parameters > ]
    [ PFSaveCTM ]
    {
        < Draw Path > | < Draw Text > | < Draw Picture >
    }
    [ PFRestoreCTM | PFResetCTM ]
    [ PFResetPageBbox | PFRestorePageBbox ]
    [ PFClearClip ]

< Set Draw Parameters > =
    [ < Change CTM > ]
    [ PFLineWidth ]
    [ PFGrayShade ]
    [ PFDeviceColour ]
    [ PFFlatness ]
    [ PFPathFast | PFPathStroked | PFPathFilled | PFPathEOFilled ]

< Change CTM > =
    {
        PFMoveCTM | PFScaleCTM | PFXScaleCTM | PFYScaleCTM | PFSetCTM
    }

< Draw Path > =
    < Draw First Subpath >
    {
```

6.1.2 Fixpoint - pt

This is actually just a representation of a decimal character which can be processed faster than standard floating points. This is done by dividing the actual representation in two parts: an integer part and a fractional part. Each part occupies two bytes in a long word. The most significant bytes are the integer part, and the least significant bytes are the fractional part, expressed as multiples of 1/65536. In binary representation, this means that there is an imaginary dot between the two words in a long word.



pt $a 0000 = 10.0
pt $6 8000 = 6.5
pt $12 3456 = 18.204437

6.2 List of Operators

6.2.1 PFAllFontCount

Action	Get the number of fonts in the fontmap. Make sure the next call to PFNextFontName will return the first font in the fontmap.
Prototype	long PFAllFontCount(Gstate GstateID, long *count);
Parameters	GstateID count : return, number of fonts in fontmap
Errors	none

6.2.2 PFCharAvailable

Action	Find out whether the given character exists in the current font.
Prototype	long PFCharAvailable(Gstate GstateID, long unicode);
Parameters	GstateID unicode : character code
Errors	ERR_ITNF (-7) : character doesn't exist in current font

6.2.7 PFCurveTo

Action	Draw a bezier curve from the current point to the endpoint using the two control points. The endpoint will become the new current point.
Prototype	long PFCurveTo(Gstate GstateID, pt cx1, pt cy1, pt cx2, pt cy2, pt x, pt y);
Parameters	GstateID cx1, cy1 : coordinates of first control point cx2, cy2 : coordinates of second control point x, y : coordinates of the endpoint
Errors	none

6.2.8 PFDeviceColour

Action	Set the colour in which should be drawn on the device. This command is only relevant if the driver knows that the device supports colour). The only driver which currently uses colour is the screen driver.
Prototype	long PFDeviceColour(Gstate GstateID, long colour);
Parameters	GstateID colour : colour to draw in
Errors	none

6.2.9 PFDriverCount

Action	Count the drivers which are available in the current configuration of PROforma. Also makes sure that the next call to PFNextDriver will return the first driver in the list.
Prototype	long PFDriverCount(Gstate GstateID, long *count);
Parameters	GstateID count : return, number of drivers
Errors	none

6.2.10 PFFlatness

Action	Set the accuracy at which curves have to be drawn. A smaller value will increase the accuracy, but will reduce the speed !
Prototype	long PFFlatness(Gstate GstateID, pt flatness);
Parameters	GstateID flatness : new value for flatness
Errors	none

6.2.11 PFFontBbox

Action	Get the maximum size of a character in the current font.
Prototype	long PFFontBbox(Gstate GstateID, pt *bbox);
Parameters	GstateID

6.2.17 PFGetFontWeight

Action	Get the weight of the current font.
Prototype	long PFGetFontWeight(Gstate GstateID, char *name);
Parameters	GstateID name : place for return string, 20 chars
Errors	ERR_ITNF (-7) : no current font

6.2.18 PFGetPageBbox

Action	Get size and origin of the current PageBbox, and the PageOrigin
Prototype	long PFGetPageBbox(Gstate GstateID, pt *result);
Parameters	GstateID result : array with six spots : xsiz, ysiz, xpos, ypos PageBbox; x, y PageOrigin
Errors	none

6.2.19 PFGrayShade

Action	The gray shade to the given percentage of black. Each device only has a limited amount of distinct grayshades. The exact number depends on the driver.
Prototype	long PFGrayShade(Gstate GstateID, pt gray);
Parameters	GstateID gray : new value for gray shade
Errors	none

6.2.20 PFInitCharPath

Action	Start a charpath for the character with given unicode. The character in the charpath will be given at the current point size, and is in the current font. The current font or the size should not be modified during a charpath loop. The standard default character for the font will be returned if the character doesn't exist.
Prototype	long PFInitCharPath(Gstate GstateID, long code);
Parameters	GstateID code : character code of character
Errors	ERR_ITNF (-7) : no current font

6.2.21 PFJustKernShow

Action	Show given string at current position with kerning. The current point will be moved to the end of the string. The width of the space characters is adjusted to make sure the text has the specified width.
Prototype	long PFJustKernShow(Gstate GstateID, short *string, pt width);

6.2.26 PFLineWidth

Action	Set the width at which stroked paths will be drawn. The linewidth has to be specified in default user space and will be influenced by the CTM ! If the linewidth is less than one pixel, the paths will be drawn hairline : a uniform width of one pixel.
Prototype	long PFLineWidth(Gstate GstateID, pt width);
Parameters	GstateID width : new value for linewidth
Errors	none

6.2.27 PFLoadFont

Action	Make sure a gstate can use the font with given name.
Prototype	long PFLoadFont(Gstate GstateID, char *fontname);
Parameters	GstateID fontname : name of font to load
Errors	any error from the QDOS i/o system

6.2.28 PFMoveTo

Action	Move the current point to the given position. If the path will be filled, the previous subpath will be closed.
Prototype	long PFMoveTo(Gstate GstateID, pt x, pt y);
Parameters	GstateID x, y : coordinates to move to
Errors	none

6.2.29 PFNextDriver

Action	Get the driverID and name of the next driver in the list of drivers. Will keep returning the data for the last driver if end of list reached.
Prototype	long PFNextDriver(Gstate GstateID, long *id, char *name);
Parameters	GstateID id : return, driverid name : return, name of driver, 60 characters
Errors	none

6.2.30 PFNextFont

Action	Get the next font in the fontlist and make it the current. The current font is not changed if there is no next font.
Prototype	long PFNextFont(Gstate GstateID);
Parameters	GstateID
Errors	none

6.2.36 PFPathDraw

Action	Draw the current path, according to the current draw parameters. If the path has to be filled, the path will be closed.
Prototype	long PFPathDraw(Gstate GstateID);
Parameters	GstateID
Errors	none

6.2.37 PFPathEOFilled

Action	Make sure the following paths are drawn filled, using the even odd rule.
Prototype	long PFPathEOFilled(Gstate GstateID);
Parameters	GstateID
Errors	none

6.2.38 PFPathFast

Action	Make sure the following paths are drawn as fast as possible. To improve the speed everything will be draw in black, without clipping and hairline. Curves may or may not use the flatness.
Prototype	long PFPathFast(Gstate GstateID);
Parameters	GstateID
Errors	none

6.2.39 PFPathFilled

Action	Make sure the following paths are drawn filled, using the winding rule.
Prototype	long PFPathFilled(Gstate GstateID);
Parameters	GstateID
Errors	none

6.2.40 PFPathStroked

Action	Make sure the following paths are drawn stroked, using the current linewidth.
Prototype	long PFPathStroked(Gstate GstateID);
Parameters	GstateID
Errors	none

6.2.41 PFRcurve

Action	Draw a bezier curve from the current point to the endpoint using the two control points. The endpoint will become the new current point.
Prototype	long PFRcurve(Gstate GstateID, pt cx1, pt cy1, pt cx2, pt cy2, pt x, pt y);
Parameters	GstateID cx1, cy1 : coordinates of first control point, relative to current point

6.2.47 PFRline

Action	Draw a line from the current point to the given position. That position will then be the new current point.
Prototype	long PFRline(Gstate GstateID, pt x, pt y);
Parameters	GstateID
	x, y : coordinates to draw a line to, relative with current point (=startpoint)
Errors	none

6.2.48 PFRmove

Action	Move the current point to the given position. If the path will be filled, the previous subpath will be closed.
Prototype	long PFRmove(Gstate GstateID, pt x, pt y);
Parameters	GstateID
	x, y : coordinates to move to, relative with current point.
Errors	none

6.2.49 PFSaveCTM

Action	Save the CTM so that it can be restored later.
Prototype	long PFSaveCTM(Gstate *GstateID);
Parameters	GstateID
Errors	ERR_IMEM (-3) : insufficient memory

6.2.50 PFScaleFont

Action	Set the display size of the current font.
Prototype	long PFScaleFont(Gstate GstateID, pt scale);
Parameters	GstateID
	scale : point size of font (relative to CTM)
Errors	ERR_ITNF (-7) : no current font

6.2.51 PFScalePage

Action	Scale the default user space by the given amount.
Prototype	long PFScalePage(Gstate GstateID, pt factor);
Parameters	GstateID
	factor : scale factor
Errors	none

6.2.52 PFScrollPage

Action	Scroll the display efficiently. This comes down to changing the PageOrigin, but part of the screen doesn't have to be redrawn. This operator is specific for interactive use, so it is only supported for screen

6.2.57 PFShow

Action	Show given string at current position. The current point will be moved to the end of the string.
Prototype	long PFShow(Gstate GstateID, short *string);
Parameters	GstateID string : unicode string to display
Errors	none

6.2.58 PFShowPage

Action	Display the current pass of the page.

Pseudo-Code	if this pass is first pass of page then init page
	display this pass
	adjust PageBbox for next pass
	if this was the last pass of the page then
	  end page
	  adjust PageBbox first first pass of next page

Prototype	long PFShowPage(Gstate GstateID);
Parameters	GstateID
Errors	any QDOS i/o error

6.2.59 PFShowPicture

Action	Display a picture (bitmap)at the current position. The picture has to be loaded in memory.
Prototype	long PFShowPicture(Gstate GstateID, long type, char *base, pt xsiz, pt ysiz, pt *colours);
Parameters	GstateID type : type of picture (bitmap) base : pointer to picture xsiz, ysiz : requested size for bitmap colours : optional array with grayshades for each colour (from white to black)
Errors	ERR_IEXP (-17) : type is invalid or wrong

6.2.60 PFTrackShow

Action	Show given string at current position. The current point will be moved to the end of the string. The inter-character spacing is adjusted to make sure the text has the specified width.
Prototype	long PFTrackShow(Gstate GstateID, short *string, pt width);
Parameters	GstateID string : unicode string to display width : width for the output
Errors	none

finetune the cache.
Operation : just type the name of the font which should be used for the test. Enter an empty line to terminate the program.

7.3 fonttest

This program was written to test the font management aspects. It asks for a command with a parameter. So type the number of the command, followed by the name of the font (for load, select and unload). There should be no characters between the command number and the parameter. Enter an empty line to terminate the program.

7.4 pagetest

This programs allows you to display a file with commands similar to the LINEdesign v1 file. Just type the (full) name of the file, or enter to end.

All commands in the command files are one character (which indicates the command), followed by some parameters (in points), separated by spaces.
Supported commands in file :

```
w : set linewidth, only one parameter
m : moveto, two coordinates, x and y
t : text, four parameters, x and y position (well almost), size, fontname
    followed by the text on the next line
P : seven parameters, first four are discarded. Other three are fill, fill
    colour and outline colour. If fill is zero then the next paths will be
stroked
    in the outline colour, else they will be filled in the fill colour (even
odd rule).
l : rline, two parameters, relative coordinates of endpoint.
c : rcurve, six parameters, relative coordinates of control points and
end point.
```

Lines should not be longer than 128 chars ! There are two example files on the disk: golfer_ldp and curve_ldp. Please note that, contrary to LINEdesign, the outline is not displayed of a filled path. Also, LINEdesign doesn't recognise the 'w' command.

7.9 cliptest

This program is identical to picttest, with the small difference that a thick curve is used as clipping path. This time it can take even longer before something changes on screen !

7.10 unicode_c

These routines are a bit different from the others. They can be used to convert unicode to the character name. It should be stated that the unicode subset we use is the same as that used by TrueType® (from Apple), and the character names are the same as used by PostScript™ (from Adobe).
There is also a table in the file (ASCII2unicode), which converts the QL character set to unicode.

Parameters	GstateID string : unicode string to display xlist : pointer to array with movements for each character
Errors	none

6.2.66 PFXYShow

Action	Show given string at current position. Each char is positioned at current position, which is then moved by the equivalent distance in the array.

Pseudo-Code	while (*string)
	{
	Display_Character(*string);
	PFRmove(GstateID, *xlist++, *ylist++);
	}

Prototype	long PFShow(Gstate GstateID, short *string, pt *xlist);
Parameters	GstateID string : unicode string to display xlist : pointer to array with horizontal movements for each character ylist : pointer to array with vertical movements for each character
Errors	none

7. Examples

While developing PROforma, we have written several small programs to test the library routines we had written. These routines are on the disk. They usually only concern a small aspect of PROforma and are quite small. Here we will not give any explanation about the actual programs, just look at the source. We will however look at what these programs do, and how they must be operated. It has to be noted that these programs do not perform much error trapping. Unfortunately not all operators are mentioned in these programs. Some of our test programs were modifications of the ones you have (and are gone), and some testing was done specifically through LINEdesign.
These programs were originally compiled using C68 v4.12. If you don not have this (or a more recent) version : get it. The compiled code is better than before, and it is the first version of C68 which also works on the Atari TT, as well as the QXL, and all other QDOS compatible systems.

7.1 test

Very small program, just draws a filled trapezoid many times. This program was used to test a simple case of the fill algorithm, PFPathFast, the gray shades and device colours. This program always draws on screen.

7.2 cachetest

This program was used to test the speed of the font cache. It can also be used as a benchmark to compare the speed of computers (when the same resolution is used), or the speed difference of the resolution. This programs shows a speed improvement of about a factor four when the cache is used, and this in the worst case. The speed improvement will often be much bigger (although this is not as easy to test). This program was also used to

7.5 showfont & showfonts

These programs display one font, or all fonts in the fontmap. When the program starts, you have to specify the driverid for the output (press enter for screen), and the device which should be used for the output. Showfonts also requires the name of the font to display (enter to end). Showfonts is the program which was used for the printout of all the fonts in the LINEdesign v2 manual. These programs are adjusted to make optimal use of the font cache.

7.6 chartest

This program can be used to display all characters in a font large. The program will see which characters are available in the given font, and display them ! The name of the font has to be specified (enter to end). You should press enter to display the next character. The character code of the current character is displayed (unicode).

7.7 charpathtest

This program (which asks for a font name and a character code, displays the requested character in outline. This is possible because the outline is built using the charpath operators. This program is very important to see how the combined characters work. Also, the fontbbox is displayed.

7.8 picttest

This program displays a bitmap on screen. You just have to specify the name of the file which should be displayed, and the type of that file. To know the file type, have a look at the PROforma_h file. Please note that it may take a while before something changes on screen (depends on the picture).

6.2.61 PFUnLoadFont

Action	Make sure a gstate can no longer use the given font.
Prototype	long PFUnLoadFont(Gstate GstateID, char *fontname);
Parameters	GstateID fontname : name of font which will no longer be used
Errors	none

6.2.62 PFUseCache

Action	Make sure the font cache is used for all text show operators until the next PFNoCache. A gstate has a default of PFUseCache.
Prototype	long PFUseCache(Gstate GstateID);
Parameters	GstateID
Errors	none

6.2.63 PFWidth

Action	Get the width (in user coordinates) of the given string.
Prototype	long PFWidth(Gstate GstateID, short *string, pt *width);
Parameters	GstateID string : unicode string width : return, width of string
Errors	none

6.2.64 PFWindowMove

Action	Adjust a gstate to represent the fact that the outline of the window has moved. This operator is only relevant for screen gstates.
Prototype	long PFWindowMove(Gstate GstateID, long dx, long dy);
Parameters	GstateID dx, dy : movement of window in pixels !
Errors	none

6.2.65 PFXShow

Action	Show given string at current position. Each char is positioned at current position, which is then moved by the equivalent distance in the array.

Pseudo-Code	while (*string)
	{
	Display character *string
	PFRmove(GstateID, *xlist++, 0);
	}

Prototype	long PFShow(Gstate GstateID, short *string, pt *xlist);

drivers.
This command sets the PageBbox to the area which has become visible (should be restore afterwards).

Prototype long PFScrollPage(Gstate GstateID, pt xdist, pt ydist);
Parameters GstateID
xdist, ydist : distance to scroll, either should be zero
Errors ERR_IEXP (-17) : scrolling only allowed in one direction at a time

6.2.53 PFSelectFont

Action Select the font with the given name as the current font.
Prototype long PFSelectFont(Gstate GstateID, char *fontname);
Parameters GstateID
fontname : name of font to select
Errors ERR_ITNF (-7) : font not loaded for this gstate

6.2.54 PFSetCTM

Action Multiply the CTM with the given transformation matrix.
Prototype long PFSetCTM(Gstate GstateID, pt xs, pt yr, pt xr, pt ys, pt xd, pt yd);
Parameters GstateID
xs, yr, xr, ys, xd, yd : transformation matrix
Errors none

6.2.55 PFSetPageBbox

Action Change the size of the PageBbox and move it. The PageOrigin is adjusted accordingly.
Pseudo-Code adjust values as requested
clip new PageBbox to imageable area
expand PageBbox by one pixel in all directions to compensate rounding errors
Prototype long PFSetPageBbox(Gstate GstateID, pt xsiz, pt ysiz, pt xorg, pt yorg);
Parameters GstateID
xsiz, ysiz : new size of PageBbox
xorg, yorg : movement of PageBbox
Errors none

6.2.56 PFSetPageOrigin

Action Move the PageOrigin by the given distance.
Prototype long PFSetPageOrigin(Gstate GstateID, pt x, pt y);
Parameters GstateID
x, y : distance to move PageOrigin in default user space
Errors none

(startpoint)
cx2, cy2 : coordinates of second control point, relative to current point (startpoint)
x, y : coordinates of the endpoint, relative to current point (startpoint)

Errors none

6.2.42 PFRemoveGstate

Action Make sure the given gstate is removed from memory.
Prototype void PFRemoveGstate(Gstate GstateID);
Parameters GstateID
Errors

6.2.43 PFResetCTM

Action Reset the CTM to the default value (no transformation).
This command also clears the CTM stack.
Prototype long PFResetCTM(Gstate *GstateID);
Parameters Gstate
Errors

6.2.44 PFResetPageBbox

Action Reset the PageBbox and PageOrigin to the initial values when the gtate was allocated.
Prototype long PFResetPageBbox(Gstate GstateID);
Parameters GstateID
Errors none

6.2.45 PFRestoreCTM

Action Restore the CTM to value that was saved before.
Prototype long PFRestoreCTM(Gstate *GstateID);
Parameters GstateID
Errors ERR_ITNF (-7) : CTM stack is empty

6.2.46 PFRestorePageBbox

Action Reset the PageBbox to the maximum size and change the PageOrigin accordingly.
Prototype long PFRestorePageBbox(Gstate GstateID);
Parameters GstateID
Errors none

6.2.31 PFNextFontName

Action	Get the name of the next font in the fontmap. The returned name will be a null string when end reached.
Prototype	long PFNextFontName(Gstate GstateID, char *name);
Parameters	GstateID name : return, name of next font in fontmap, 60 chars
Errors	none

6.2.32 PFNoCache

Action	Make sure the font cache is not used for all text show operators until the next PFUseCache.
Prototype	long PFNoCache(Gstate GstateID);
Parameters	GstateID
Errors	none

6.2.33 PFNrOfCopies

Action	Set the number of identical copies of each page which should be reproduced by the device. This routine is only supported by some devices which have this built in feature (i.e. some laser printers).
Prototype	long PFNrOfCopies(Gstate GstateID, long count);
Parameters	GstateID count : number of identical copies
Errors	ERR_IPAR (-15) : driver doesn't support this

6.2.34 PFPathClear

Action	Clear the current path.
Prototype	long PFPathClear(Gstate GstateID);
Parameters	GstateID
Errors	ERR_IMEM (-3) : insufficient memory

6.2.35 PFPathClip

Action	Create a new clipping path, built from the current path, clipped by the old clipping path.
Prototype	long PFPathClip(Gstate GstateID);
Parameters	GstateID;
Errors	ERR_IMEM (-3) : insufficient memory

Parameters	GstateID string : unicode string to display width : width for the output
Errors	none

6.2.22 PFJustShow

Action	Show given string at current position. The current point will be moved to the end of the string. The width of the space characters is adjusted to make sure the text has the specified width.
Prototype	long PFJustShow(Gstate GstateID, short *string, pt width);
Parameters	GstateID string : unicode string to display width : width for the output
Errors	none

6.2.23 PFKernShow

Action	Show given string at current position with kerning. The current point will be moved to the end of the string.
Prototype	long PFKernShow(Gstate GstateID, short *string);
Parameters	GstateID string : unicode string to display
Errors	none

6.2.24 PFKernWidth

Action	Get the width (in user coordinates) of the given string using kerning.
Prototype	long PFKernWidth(Gstate GstateID, short *string, pt *width);
Parameters	GstateID string : unicode string width : return, width of string
Errors	none

6.2.25 PFLineTo

Action	Draw a line from the current point to the given position. That position will then be the new current point.
Prototype	long PFLineTo(Gstate GstateID, pt x, pt y);
Parameters	GstateID x, y : coordinates to draw a line to
Errors	none

bbox : return, array with 4 values
bbox[0], bbox[1] : size of box
bbox[2], bbox[3] : offset of character origin in bbox

Errors ERR_ITNF (-7) : no current font

6.2.12 PFFontCount

Action Get the number of fonts in the fontlist.
Make sure the current font is the first font in the fontlist.

Prototype long PFFontCount(Gstate GstateID, long *count);

Parameters GstateID
count : return, number of fonts in fontlist

Errors none

6.2.13 PFGetFontFamily

Action Get the family name of the current font.

Prototype long PFGetFontFamily(Gstate GstateID, char *name);

Parameters GstateID
name : place for return string, 60 chars

Errors ERR_ITNF (-7) : no current font

6.2.14 PFGetFontName

Action Get the name of the current font.

Prototype long PFGetFontName(Gstate GstateID, char *name);

Parameters GstateID
name : place for return string, 60 chars

Errors ERR_ITNF (-7) : no current font

6.2.15 PFGetFontNotice

Action Get the notice of the current font.

Prototype long PFGetFontNotice(Gstate GstateID, char *name);

Parameters GstateID
name : place for return string, 80 chars

Errors ERR_ITNF (-7) : no current font

6.2.16 PFGetFontVersion

Action Get the version of the current font.

Prototype long PFGetFontVersion(Gstate GstateID, char *name);

Parameters GstateID
name : place for return string, 10 chars

Errors ERR_ITNF (-7) : no current font

6.2.3 PFCharPathEl

Action	Get the command in the charpath initiated by PFInitCharPath.
Prototype	long PFCharPathEl(Gstate GstateID, short *comm, pt *x1, pt *y1, pt *x2, pt *y2, pt *x3, pt *y3);
Parameters	GstateID comm : return command (rmove, rline, rcurve, combined, end) comm==rmove x1, y1 : distance for relative move comm==rline x1, y1 : distance for relative line comm==rcurve x1, y1 : relative coordinates of first control point x2, y2 : relative coordinates of second control point x3, y3 : relative coordinates of endpoint comm==combined x1, y1 : distance for relative move x2 : short2pt(unicode) of base character y2 : short2pt(unicode) of accent character
Errors	none

6.2.4 PFClearClip

Action	Clear the current clipping path.
Prototype	long PFClearClip(Gstate GstateID);
Parameters	GstateID
Errors	ERR_IMEM (-3) : insufficient memory

6.2.5 PFClearPage

Action	Clear the page (inside PageBbox only).
Prototype	long PFClearPage(Gstate GstateID);
Parameters	GstateID
Errors	none

6.2.6 PFClosePath

Action	Close the last subpath in the current path.
Pseudo-Code	Add a line from the current position to the start coordinates of the path.
Prototype	long PFClosePath(Gstate GstateID);
Parameters	GstateID
Errors	none

6. Operators

As has been mentioned before, PROforma is actually an extension thing. This means that PROforma is actually implemented as an extension of the operating system.

6.1 Parameters

However, as PROforma was originally developed as a library for LINEdesign, it is designed in such a way to be called efficiently from C (well C68 to be precise). This means that parameters are actually passed on the stack. All parameters are four bytes long and contain either an integer (long word), a pointer to a string, or a fixpoint number.
Although we have only supplied a C68 interface, writing interfaces for other languages is not difficult. However PROforma should never be called from interpreted SuperBASIC. PROforma is written in C and this means that it uses the stack for local data. However, in interpreted SuperBASIC the stack can suddenly move, and PROforma can't handle this !

6.1.1 Strings

Contrary to the approach which is usually taken by the operating system, we use null terminated strings (instead of preceding them by the length).
Another problem is that we use two kinds of strings. We have the C standard null terminated strings which contain ASCII codes (one byte per character), and we have unicode strings, in which each character is represented by a word (2 bytes). The use of unicode allows us to have character sets which contain more than 256 characters, and have all characters at a fixed spot. Most general characters (about codes 32 to 126) are at about the same position as in the ASCII character set.

```
        [ < Draw Subpath > ]
    }
    PFPathDraw | PFPathClip | PFPathClear

< Draw First Subpath > =
    PFMoveTo
    {
        PFLineTo | PFRline | PFCurveTo | PFRcurve
    }
    [ PFClosePath ]

< Draw Subpath > =
    [ PFMoveTo | PFRmove ]
    {
        PFLineTo | PFRline | PFCurveTo | PFRcurve
    }
    [ PFClosePath ]

< Draw Text > =
    [ PFLoadFont ]
    [ < Set Text Parameters > ]
    [ PFUseCache ]
    PFMoveTo
    {
        PFShow | PFKernShow | PFXshow | PFXYShow | PFTrackShow | PFJustShow |
PFJustKernShow
    }
    [ PFUnloadFont ]
    [ PFNoCache ]

< Set Text Parameters > =
    [ PFSelectFont ]
    [ PFScaleFont ]

< Draw Picture > =
    PFMoveTo
    PFShowPicture
```

5. PROforma sessions

If you want to use PROforma, there are certain rules you have to follow to assure correct operation. The most important rules concern the order in which the operators have to be used. For instance, you should not change the transformation matrix during the building and drawing of a path. This is done because PROforma is built for efficiently. Everything has to be fast and flexible. This unfortunately means that robustness is not one of PROforma's strong points. Doing unexpected things may cause unexpected results.

To try to make things easier we will now give an outline of a common procedure for PROgrams which generate their output using PROforma. That is programs for programs which are not interactive. A program which is interactive does not produce pages as suggested here. Generally speaking, the scheme will be very similar though.
The scheme listed here doesn't include two groups of operators. The operators which request information from the system, as these can be used between drawing commands, as they don't change anything in the system. Also some commands like PFScrollPage and PFWindowMove are not listed. These commands can also be used at about any moment.

There are some general notation used in the following scheme :

command
: Execute the command.

{ command }
: The command can be repeated zero or more times.

[command]
: The command can be executed at most once.

command1 | command2
: Either command1 or command2 will be executed.

< command >
: Composite command, elaborated somewhere else in the scheme.

Of course these rules can be combined to form a more elaborate scheme.

```
< PROforma session > =
   PFInitGstate
   [ PFNrOfCopies ]
   /* FOR each page */
   {
```

4.6 Charpaths

For drawing programs like LINEdesign it is interesting to be able to extract the outline of a character from a font as this allows the user to make some individual changes to the characters(s). Therefore this feature is supported by PROforma. And consists of a loop like :

```
select character which should be converted
DO
    get the next path operator
    process it
UNTIL end of character
```

Note: the PFInitCharPath and PFCharPathEl commands should only be used in such a loop, just as all the other iterators in PROforma.

PFInitCharPath

Select the character for which the outline will be extracted. Selects that character from the current font at the current size of that font.

PFCharPathEl

Get the next path element for that character. This can be a move, line, curve or end of character command (which can be interpreted as a PFPathDraw).

4.3 Available Fonts

Information about all the available fonts. This allows the client to know which fonts are loaded and can be used.

PFFontCount
: Get the number of fonts which are loaded (in the fontlist).

PFNextFont
: Select the next font in the list of fonts as the current font. If the last font in the list was the current, the first font will be selected as next. You can then use the commands to get information about the the current font.

And you can enquire about all the fonts which are in the fontmap.

PFAllFontCount
: Get the number of fonts in the fontmap. Also makes sure that the next enquiry with PFNextFontName will return the first font in the font map.

PFNextFontName
: Get the name of the next font in the fontmap.

4.4 Text display

Routines to display a font. All these commands use UniCode character codes. Strings are similar to C strings, null terminated. All routines display the requested character(s) if it exists, or the first character in the list if it doesn't exist (normally the .notdef character), or nothing at all (if the first character in the font is not .notdef).

PFShowChar
: Show the requested character at the current position. The current position doesn't change.

PFShow
: Show the requested string at the current position. Characters are placed proportional. The current position will now be the place where the next character should be placed.

PFKernShow
: Show the requested string at the current position with kerning. The current position will now be the place where the next character should be placed.

4. Font Management

A very important part of PROforma concerns the manipulation and displaying of vector fonts. The vector fonts which are used in PROforma are a direct descendant of the Adobe Type I font format, but optimised for efficient access, and low memory consumption. A set of utility programs to convert Type I (.pfb) fonts into PROforma (_pff) fonts exists (which works in most cases).

4.1 Font Loading

Fonts have to be loaded upon request of the client, and can also be released from memory by the client. Fonts only take as much memory as they occupy on disk. Fonts are always referenced by their name. Font names are case dependant ! If a font is loaded by several gstates, it is only kept once in memory. So a font is only released from if there is at least one gstate which has that font loaded. A font is always removed from memory when there are no more gstates which have loaded that font.

PFLoadFont

Load the given fontfile into memory, to make that font available to the client. This command will automatically select the just loaded font and scale it to one point. This command needs a fontname. The corresponding filename be searched in the fontmap. So only fonts which are in the fontmap can be used. Font files are searched 'as is', and on the configured path e.g. 'win1_fonts_;flp1_' will search 'font_pff' first on 'win1_fonts_font_pff' and later (if not found) on 'flp1_font_pff'.

PFUnLoadFont

Stop using the given font. After UnLoad the current font is no longer defined. If the font is not loaded by any other Gstate, then the memory which this font occupies will be released. This actually allows PROforma to release the font. The client can no longer use that font, unless it is loaded again.

PFSelectFont

values are given in default user space.

PFWindowMove

This command adjust the internal structures to a Gstate to match the position of the screen window (it does nothing for non-screen Gstates). This should be called if the owning job uses the Pointer Environment and has been replaced. So this command should be called after a iop.outl or wm.chwin call.

3.8 Controlling the page

PFShowPage

Copy the device buffer to the device. Will display the (part of) the page which has already been rendered. If the page will be built in several passes, this routine automatically adjusts the internal structures to render the next pass. The client is responsible for rebuilding the page. If this was the last pass of a page, the internal structures will be adjusted for the first pass again. This command also allows multiple copies of the page to be produced. However this option will only work if the device supports this (like a laser printer). The actual image in the buffer is not cleared by this command.

PFScrollPage

Allows the client to efficiently scroll in interactive applications. Therefore this command only does something in screen devices. It is only allowed to scroll in one direction at a time (horizontal or vertical). The PageBbox and PageOrigin are automatically adjusted to fit the newly visible space.

PFClearPage

Clear the internal buffer for the device. This will only clear the area which falls inside the current PageBbox, as this allows efficient redrawing of the screen for interactive applications.

Some devices (usually laser printers) can easily produce following copies after the first one at great speed. This is supported by PROforma.

PFNrOfCopies

Set the number of copies which should be produced. Only works if the device supports it (and the driver of course). If not supported this command returns an error.

PFRline

Construct a line from the current point with the given displacements. After this command, the endpoint will be the new current point.

PFCurveTo

Construct a bezier curve from the current point to the given endpoint, using the given control points for direction (all absolute coordinates). After this command, the endpoint will be the new current point.

PFRcurve

Construct a bezier curve from the current point to the given endpoint, using the given control points for direction (all relative coordinates). After this command, the endpoint will be the new current point.

PFClosePath

Make sure the current subpath is closed. A line segment will be added from the end of the subpath (the current point) to the beginning.

PFPathDraw

Make sure the path which was built is actually rendered in the buffer. The path will be empty after this command, and the current point reset.

PFPathClear

Clear the current path (make it empty). This also resets the current point.

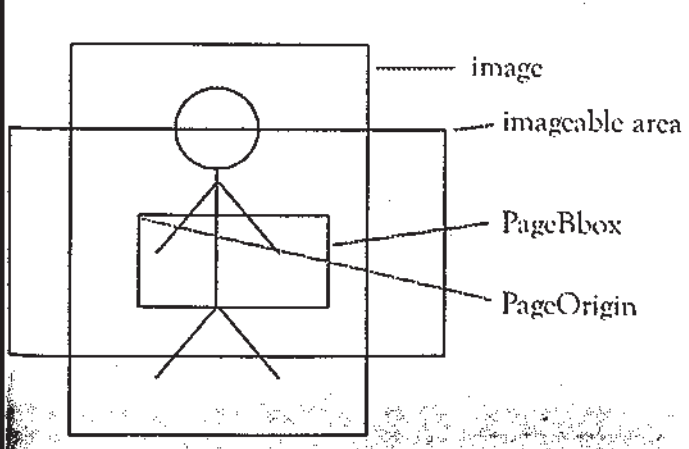
3.7 Controlling the visible area

PFResetPageBbox

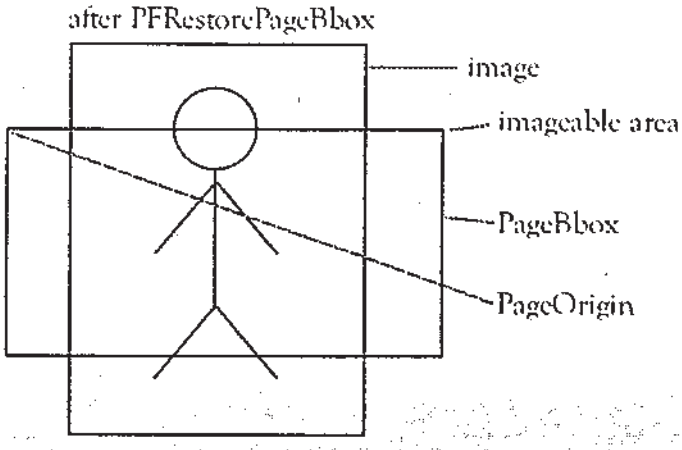
Reset the PageBbox and PageOrigin to the original values when the Gstate was initialised (so reset it to the entire visible page).

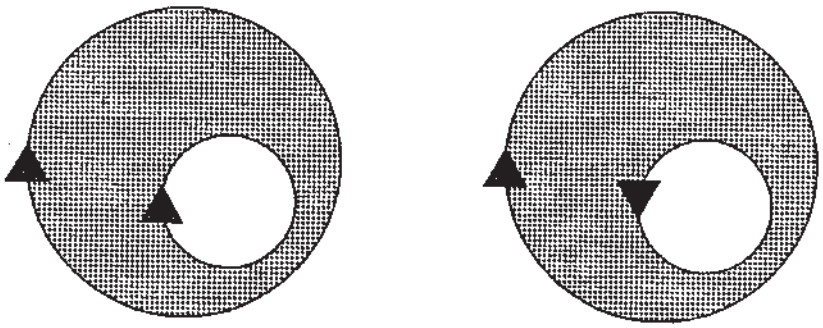
PFRestorePageBbox

Reset the PageBbox to the entire visible page, and set the PageOrigin to the point which is already at the top left of this area. This allows the client to restore the PageBbox after a PFScrollPage of PFSetPageBbox command.



after PFRestorePageBbox





To determine whether an area is filled by the even odd rule. Initialise the winding counter to zero and draw a line to infinity. For every edge which is crossed you should add one to the winding counter. If the resulting winding counter is odd, then the area will be filled. All areas should be closed for this rule to work.

PFPathFast

Notify PROforma that all future path construction commands will be performed as fast as possible. This means that everything will be drawn hairline (pixel width), in black (so no grayshades), and without clipping.

PFDeviceColour

Because PROforma doesn't support true colour yet, but the QL screen does, there is a temporary command included to set the device colour. This command only affects screen devices, and allows the client to set the drawing colour of the other commands. It only accepts four colours, to support the standard QL mode 4 screen.

PFGrayShade

Select the current grayshade for drawing. Grayshades are given in percentages. All devices have a few distinct grayshades. Higher resolution devices have more grayshades than low resolution devices.

PFLineWidth

Set the linewidth in user coordinates for stroking. All lines and curves which are drawn stroked after this command is given will have the given linewidth until the next PFLineWidth command. The linewidth should not be changed while the path is built as this could cause unexpected results.

PFFlatness

Set the flatness. This is the amount of tolerance the approximation of the bezier curves allows. A high value increases drawing speed, but decreases the accuracy. If this value is too high, curves will degenerate to polygons.

Note that this parameter does not necessarily influence the behaviour of curves which are drawn 'fast'. They may be drawn with a special routine which is faster.

large flatness

small flatness

PFScaleCTM

Scale the current transformation matrix. The origin remains in the same position. All following objects are enlarged by the given factor. This command maintains the ratio, so everything is scaled by an equal amount in all directions. This routine is a macro for PFSetCTM.

PFXScaleCTM

Scale the current transformation matrix along the x axis. The origin remains in the same position. All following objects are enlarged along the x axis by the given factor. This command does not maintain the ratio, scaling is only along the x axis (which can be rotated) ! This routine is a macro for PFSetCTM.

PFYScaleCTM

Scale the current transformation matrix along the y axis. The origin remains in the same position. All following object are enlarged along the y axis by the given factor. This command does not maintain the ratio, scaling is only along the y axis (which can be rotated) ! This routine is a macro for PFSetCTM.

PFSetCTM

Set the current transformation matrix to the given value, which is relative to the previous value of the CTM. This command should not be performed during the building of a path as that would give problems when closing the current subpath (just as the other commands which change the CTM).

PFResetCTM

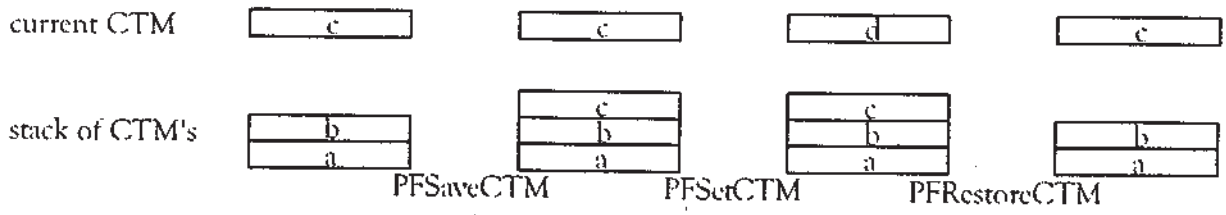
Reset the current transformation matrix to the standard values, being all measurements in default user space (initially points, with 72points/inch, as there is no real standard for points, abbreviated pt), and the origin at the top left, and axisses extending right and down. This command also clears the list of CTM's which are saved. However, default user space can be changed with PFScalePage.

PFSaveCTM

This command allows the client to save the values of the CTM, so that it can later be recovered. Can be called as often as needed (memory permitting, not that it takes much memory).

PFRestoreCTM

Restore the CTM to a previously saved CTM. You should consider the list of CTM's as as last in, first out (LIFO) stack. PFRestoreCTM removes the last topmost CTM from the stack and makes it the current.



3. Graphics

This chapter gives an review of what is possible with PROforma when you do not want to use text. Some of these things also have their consequences when displaying text, but we have chosen to make the font management a separate chapter as there is so much to say about it.

3.1 Gstate

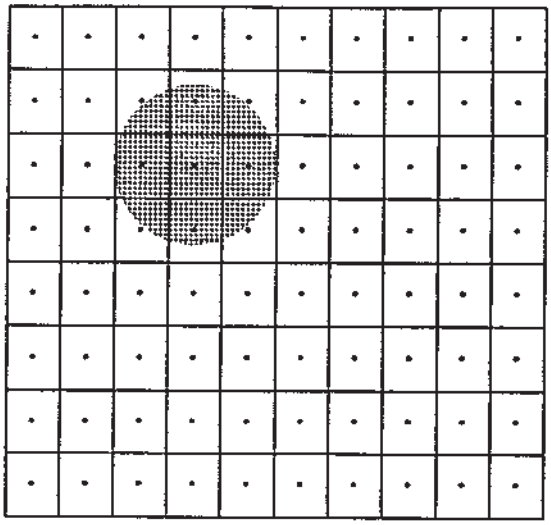
All commands in PROforma actually require a gstate as an access point to PROforma. Therefore we have introduced some commands to create and delete gstates.

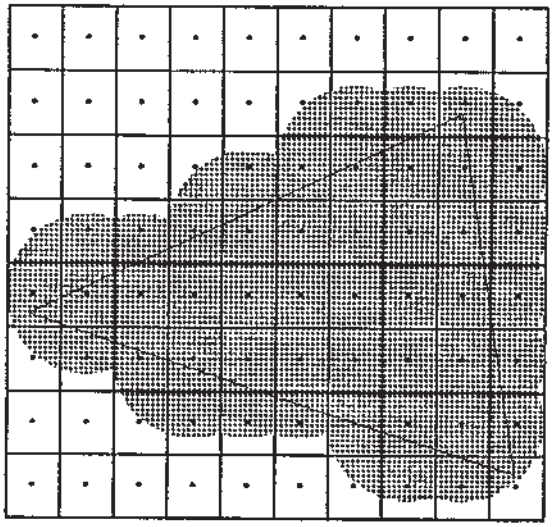
PFInitGstate

Create a new gstate. The client has to specify which driver to use and the device which should be used for that driver. The client also determines the size of the page and gets to know how many passes are necessary to render a complete page. Drivers are always indicated by a driverid. This is an integer which identifies a device. Driverid's are not fixed, they can differ between versions and configurations of PROforma. Some drivers need to have a device specified as well. Other drivers are only capable of sending their image to one device and always send it to that one. You can always open a screen gstate by specifying zero as driverid (this is the only fixed driverid). The screen driver currently has a maximum size of 720x540pt. This is approximately a 12inch screen (10x7.5 inch).

The client can specify the requested page size and position on the device. However, this area is clipped to the actually usable area. If the resulting area is non-existing (so no part of the requested area is usable on that printer), an error is reported.

When a gstate is opened, PROforma assumes you want to use the entire page. So the PageOrigin is set to the position of the top left pixel of the PageBbox on the device.





Another problem often encountered in dot matrix printers is banding. This means that there is a regular repetition of lighter and darker horizontal bands. This is mainly caused by the use of ink ribbons. They are also used for printing text and therefore the area in the middle of the ribbon is used more than the top or bottom. The less used area produces darker dots. On the other hand the ribbon also rotates horizontally, and this may also cause a difference in darkness (some parts were used more than others).

2.2.5 inkjet or bubblejet printer

This is generally speaking the same as a dot matrix printer. However, the ink is fluid now, and it is usually absorbed by the paper. This causes an additional problem as the size of the dots now also depends on the type of paper. The shape of each dot can also change, and this also depends on the paper (very local). Inkjet or bubblejet printer usually suffer a lot less of banding. A major advantage of inkjet printers is that they are very good at filling black regions, although the paper may bend because of the wet ink.

2.2.6 laser printer

Laser printers either draw their page in black (most often) or in white (as copiers do). This has certain effects on the result (making it either darker or lighter), and pixels don't always have the same size (especially in corners, this is sometimes corrected or used by the printer (so called resolution enhancement). Because of the technology used (toner which sticks to charged particles) laser printers have got problems with small (or thin) areas (like hairline paths, which fade away), and with large black areas (which become lighter in the middle). On the other hand, laser printer have the highest real resolution (smallest dots), and gives the highest quality output. Actually, a 300 dpi laser printer giver better, crisper output than a 300 or 360 dpi dot matrix or inkjet printer.

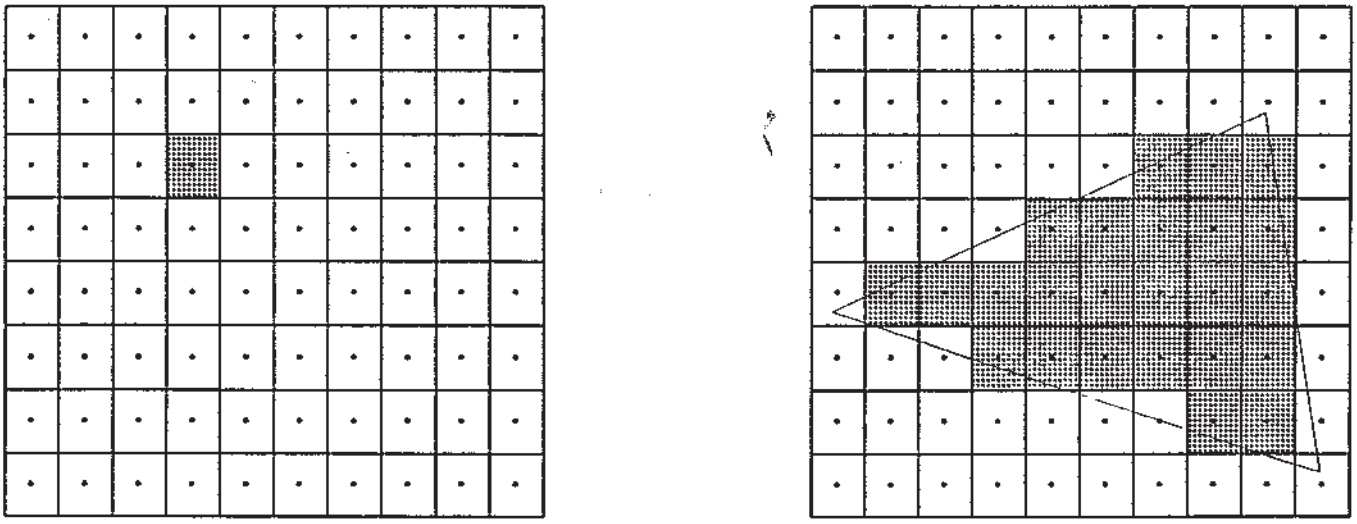
When transferring the buffer to the printer, PROforma immediately makes sure the buffer is ready for the next pass of that page, or, if this was the last pass of the page, it makes sure the buffer is ready for the first pass of the next page. The buffer is however not cleared. This is done to allow small changes to be made in the buffer without redrawing all the other stuff (which is only relevant if the page is produced in one pass and can be particularly useful for interactive use and mailmerging).

The buffered approach is actually taken one step further in PROforma. It also applies to paths. Although all parameter about how to draw the path have to be known in advance, the path is not actually drawn while it is built. The path is only drawn when you call a command to do so.

To be 100% correct, we must state that some device drivers (possibly in some versions and with some parameters) may actually bypass the buffer(s). However, this can only explain some 'unexpected' behaviour in some cases. It should never be assumed.

2.2.2 Pixels

PROforma has it's own convention on pixels. It assumes that pixels are rectangles, and that they are positioned between the grid lines.



In this picture you see the grid lines, the pixel centres, and the actual pixels. On the right, there is a filled triangle drawn. As you can see, pixels are only drawn when the centre lies inside the triangle. A boundary situation occurs when the edge of the triangle coincides with a pixel centre. In this case the edge is shifted to the right over in infinitely small amount.

This also means that areas which have a thickness of less than a pixel may be (partly) invisible if no pixel centres fall inside the path.

2.1.16 Font Caching

To increase the drawing speed of text, often used characters are also kept in an internal format which can be displayed much faster than the standard representation on the font. This is called the font cache. There are two limitations imposed by the font cache. The font cache is not capable to display fonts with clipping. Only characters which are not slanted or rotated (so only scaled) can be handled by the font cache. This actually means that some fonts can never be cached (fonts which are internally slanted or rotated). The font cache is also not used for the characters which are partly invisible.
Because the font cache has a limited size, a replacement algorithm must be used. In the case of PROforma, we make sure that only the least recently used characters are removed from the font cache. PROforma makes sure that the capacity of the font cache is not reduced because of fragmentation.

unfortunately, the font cache doesn't use a magic trick. Although a cached characters draw at least four times faster than a character which was not cached, you can only gain speed if the character which is cached is used again before it is removed. So if you now in advance that a certain character will only be displayed once, switch off the cache ! This should be done because actually placing a character in the font cache can be hard work !

2.1.17 Extended Character Set

Because typography uses many characters, PROforma uses a special extended character set, which contains much more characters then the standard character set which is supported by the operating system.
All character strings which are used to display text use an extended character set, unicode. In unicode all characters are a word long (two bytes instead of one).
Actually unicode is a character encoding, while PROforma needs a glyph encoding.This means that some things are not supported by unicode which PROforma needs and vice versa (e.g. ligatures). So PROforma uses only a subset with some small changes (actually the same character set as used in TrueType).

2.1.18 Kerning

To increase the cohesion of a combination of characters, it is often not enough to position all characters side by side, put some character combinations have to be put closer together (or further apart) to make sure that they are visually equally spaced (same amount of whitespace between characters). This process is called kerning. A typical example is the word "AWAY."

2.1.9 User space

User space is the coordinate system which is used to tell PROforma where and how to display path or text objects. The user space is converted into default user space by PROforma. This user space divides an inch in 72 equal parts, the axisses are horizontal (x) and vertical (y). The origin is at the top left, and the axisses extend right and down. The unit of 1/72 inch is called a point (pt). Note that a point can be defined slightly different depending on the source: some say there are 72.27 points in an inch, others say 72.307 points per inch.
Please note that PROforma allows to scale the default user space. This would allow the user to specify all coordinates in inches, or centimetres,...

2.1.10 Device space

Internally, PROforma transforms all coordinates from user space to device space. This resembles the position of the picture elements (pixels) of the device. Thus PROforma can decide which pixels to turn on or off.

2.1.11 Current point

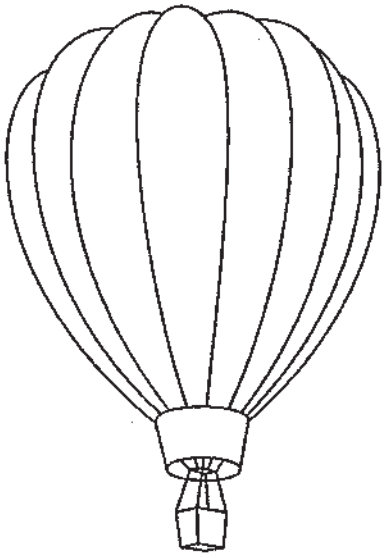
The current point is very important when building a path. All path construction commands start at the current point, and set the current point to their endpoint.
It is also the start position on the baseline for text, and set to the end of the text. And it is also the position where a bitmap can be placed.
However, the current point is not always stable. For instance, the current point can not handle changes in the CTM. To avoid this kind of problems, see "PROforma sessions."

2.1.12 PageBbox & PageOrigin

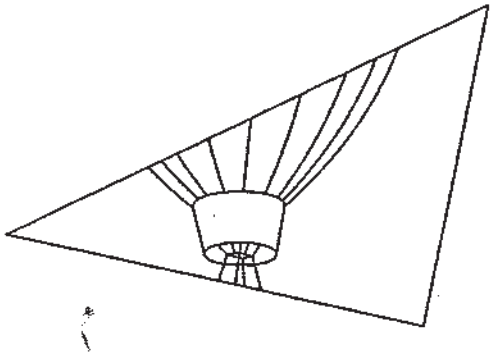
PROforma has a special view on how things have to be visualised on the chosen device. For starters there is the "page." This entity contains all path and text objects which have to be drawn. The actual image of the object depends on the CTM.
On the other hand, the page has to be visualised on the chosen device. Two things are important for this, the PageBbox which gives the origin and size (on the device) where the page (or part of) will be visualised. Which part of the page will be shown is determined by the PageOrigin, which is the coordinate (in default user space) of the point in the top-left corner of the PageBbox.

2.1.7 Clipping path

A clipping path is a special path which is not actually drawn, but which is used as a mask for all drawing operation (except text when the cache is used, see later). So the path itself is not drawn, but instead only the places which would be coloured by drawing the path are candidates for all future drawings until the clipping path is cleared.



left : without clipping path
under : with triangle as clipping path

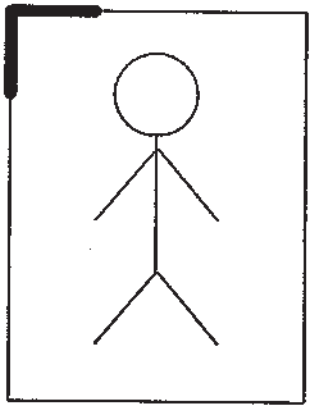


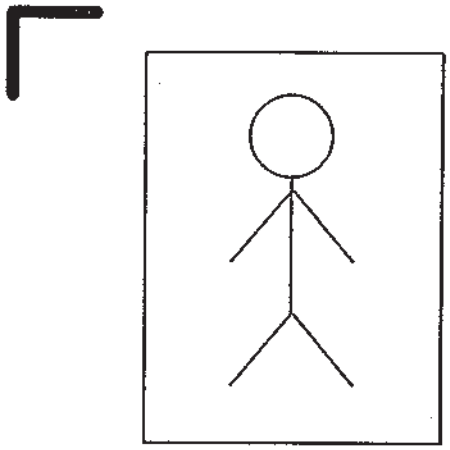
2.1.8 Transformation matrix

This is a structure (actually a matrix), which explains how the coordinates which are passed to PROforma will be transformed to default user space.

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ \text{x-dist} & \text{y-dist} \end{pmatrix}$$

x-dist = 50
y-dist = 15





2. PROforma

2.1 Concepts

2.1.1 Graphics State - Gstate

All operations in PROforma need some kind of entry point, just to let PROforma know which device has to be used, what parameters are currently valid, how big the drawing board is etc. To prevent the client (that is the user, or the application which wants to use PROforma) from having to pass these details with every command, with all possibilities of mistakes, these parameters are combined into a general, internal structure for PROforma. This structure is a 'graphics state' or 'Gstate'. A gstate contains information about :

- the device which is used (screen, printer) and the size of the usable area, specification about how to draw on that device,...
- All parameters about the current drawing methods like gray shade, line thickness,...
- All information about the fontlist, current font,...
- The current transformation matrix (CTM), list of saved CTM's, current point,...
- All information about the current state of the iterators like device iterator, fontmap iterator, charpath iterator,...
- Information about the current clipping path.

2.1.2 Driver & Device

A driver is a set of characteristics and routines which describe the behaviour of a certain output device (like a printer, or the screen). This usually includes details as size, resolution, available colours... On the other hand you can probably attache your printer both to a serial port, or a parallel port, or maybe you just want your image to to output to a file. Therefore, you always have to specify the driver (how to draw), and a device (where to draw) when you allocate a gstate.

PROforma will still be unable to load the font. The program will not stop you to try to include the same file in the fontmap twice.
Fontfiles. All references inside PROforma to a font are done using the name of that font. However this is usually not the same as the name of the font file. Therefore, the fontmap makes sure PROforma knows which files to load when a font is requested (the fontmap) and where to search for that file (the search path).
When you add a font to the fontmap, the PFConfig program tries to distinguish between the font name and directory, as we did not want to assume that all users use directories. Thus we assume that the actual filename does not contain underscores ('_'). For instance 'win1_pf_dixon_pff' is broken into three pieces: the directory 'win1_pf_', the filename 'dixon', and the extension '_pff'.

• Add all fonts in directory to fontmap. You can add an entire directory with fonts to the fontmap with this one. Again this can create duplicates. This command is particularly useful when you have acquired a disk full of PROforma fonts from some source, and you want to be able to use them.

• Delete a font from fontmap. This command allows you to select a font you want removed from the fontlist. You get a sorted list of all fonts which are currently in the fontmap, and if you indicate a font from the list, it is removed. Because the fonts are sorted (by name, case dependant), you can easily throw out duplicates.

the use of high quality colour printers, and that users can automatically get colour separations. Also, we hope to add dashed lines, and some variations on line caps, line joins and maybe even calligraphic lines.

Also, we are going to continue to develop PROforma on more powerful systems. For instance, we might let PROforma use the blitter on those systems which have one (like the Atari Mega ST, if the OS lets us). We also foresee special versions to make optimal use of 68030 or higher processors, and maybe (depends on our compiler support routines), also for floating point co-processors.

1.4 This manual

This manual has been produced using a specialised in-house program to print text using the PROforma software, which can include pictures. The text was printed on an Atari SLM804 laser printer using the Grand Old Style font family. The pages were rendered by an Atari TT under QDOS emulation (level E drivers). The pictures were all created in LINEdesign.

1.5 Installation

PROforma has the actual appearance of a normal job (as most application programs). This means that PROforma can be loaded with a line like :

```
EX PROforma
```

The fact that PROforma has the form of a job (and not a resident extension as most libraries like the Menu Extensions), has certain advantages. Jobs can always be loaded (if you have enough memory), and jobs can always be removed. So if you want to release the memory which is used by PROforma, you can just remove the job. Of course the disadvantage of this scheme is that you can accidentally remove the PROforma job, which is dangerous as all programs which use PROforma will also be removed, so you could loose data that way.

WARNING ! Removing the PROforma job may result in data lost by the applications which are at that moment using PROforma.

When PROforma is loaded, it will automatically search for the 'PFfontmap' file. This file contains configuration information about the fonts which can be used, the search path for fonts, and the memory usage of PROforma. Normally, the PFfontmap file is searched on the program default device (cfr. the Toolkit II command PROG_USE). However, if you

1. Introduction

1.1 What is PROforma ?

PROforma is short for 'PROGS Font & Raster Manager', and it does exactly what this name suggest. It is a library of routines to manage and display vector graphics and fonts on (raster) devices like screens and printers.
The availability of a separate program to manage graphics and fonts has several advantages. It allows application developers to create output of equal quality (resolution permitting) on several devices, and they can share resources. In short this means that the PROforma library only has to be loaded once, independent of the number of applications which use it. Also fonts only have to be loaded once, and can be shared between applications.
PROforma was originally developed as the graphics library for LINEdesign. That does not mean that this is the only kind of application for which PROforma is of use. PROforma is perfectly suitable as well for desktop publishers, word processors, business graphics and all applications which want high quality output (which must be just about every application except compilers and games). Actually, even at the time of writing there are things which are possible with PROforma and can't be accessed through LINEdesign.
As a library, PROforma has the form of an extension thing (if you don't know what that is, don't worry). It is specifically designed to be called efficiently from C68. However, it is quite easy to access PROforma from any other programming language.

1.2 Disclaimer & Copyrights