filePro PDF printing

filePro version 5.7.03 introduces the ability to print directly to a PDF document. There are two parts to this feature – specifying the destination, and the new FPML "filePro markup language".

Specifying the destination

At its most basic, PDF printing is accomplished by specifying "PDF:filename" as the output destination. This will cause the output to be sent to the specified filename in PDF format. Note, however, that any print codes must be in FPML format, as filePro will not translate other print code languages into PDF.

There are several options for the "PDF:destination" specification:

- `PDF:filename`
  Sends the output to the specified filename.
- `PDF:>filename`
  Sends the output to the specified filename.
- `PDF:|command`
  Sends the output to the standard input stream of the specified command.

On Windows platforms, there are several additional options available:

- `PDF:[open]`
  Creates a PDF document in the user's "temp" directory, and starts the default PDF application to display it.
- `PDF:[print]`
  Creates a PDF document in the user's "temp" directory, and sends it to the default printer.
- `PDF:[printto]printer`
  Creates a PDF document in the user's "temp" directory, and sends it to the specified printer.
- `PDF:[edit]`
  Creates a PDF document in the user's "temp" directory, and starts the application defined as the PDF editor.

Note that the "`[print]`" and "`[printto]`" options will work with "Windows-only" or "host-based" printers, if the printer's device driver is installed.

FPML – The filePro Markup Language

In order to generate anything beyond "plain text" output, you need to use an XML-like "markup language" called FPML – the "filePro Markup Language" – to control things such as text formatting and image processing.

For many simple reports and forms, a PDF file can be generated by simply changing the print code table to "fpml", and specifying a PDF destination as described above. For more complex print jobs, you may need to include FPML in your output.

There is a new option on dmoedef's F8/Options screen – allow embedded FPML on form – which, if

set, allows you to put FPML print codes directly on the form and in data included on the form. Note, however, that this means that data cannot contain the "<" character, as this will be interpreted as the start of an FPML tag. With this option off, it is still possible to embed FPML on the form and in data, by prepending an ESCape character immediately before the "<". For example:

```
xx = chr("27") & "<font color='red'>"
```

Unless otherwise noted, everything is case-insensitive. For example, these are equivalent:

```
X="CENTER"
X="Center"
x="center"
```

Values can be enclosed in single- or double-quotes:

```
FILE="foo.jpg"
file='foo.jpg'
```

Unknown tags are silently ignored.

The "fpml" print code table

filePro includes a new "fpml" print code table. This is based on the PCL print code tables already included with filePro. Any output format which uses a filePro-supplied PCL print code table will likely work as-is by simply specifying "fpml" as the printer type, and giving a PDF destination. (Note that PCL codes generated in processing will not work as-is.)

Format

The basic format for FPML tags is:

```
<TAGNAME ATTRIBUTE="VALUE" ATTRIBUTE="VALUE" … >
```

Note that, unlike XML, there are no "close" tags, nor do the tags end with a slash.

For example, to use the FONT tag to set the font to 18-point bold, you would use:

```
<FONT SIZE="18" BOLD="ON">
```

Remember, case is not significant, and you can use either single or double-quotes, so the following is equivalent:

```
<font size='18' bold='on'>
```

Note that numeric attributes can include decimals. For example:

```
<font size="14.4">
```

Page position

When specifying a page position, the coordinates start at (0,0) as the upper-left corner, within the margins. X increases to the right, and Y increases down the page. Coordinates are specified in "points", which is the typography measure for 1/72 of an inch.

<NUL>

A no-op. This does nothing, and any attribute/values that are given are ignored. This is useful in places where you build the tag at runtime, and may need to have a placeholder when nothing needs to be done.

<PAGE>

Sets the page size and orientation. Note that, if data has already been sent to the current page, this will implicitly start a new page.

All attributes are optional.

SIZE Specifies the page size

- LETTER
- LEGAL
- A3
- A4
- A5
- B4
- B5
- EXECUTIVE
- 4x6
- 4x8
- 5x7

ORIENTATION Specifies the page orientation

- LANDSCAPE
- PORTRAIT
- Alternatively, an explicit height and width can be specified using HEIGHT and WIDTH.

The default is LETTER, PORTRAIT. This setting will be retained until explicitly changed.

Margins are currently fixed at 1/4 inch.

Example:

```
<page size="letter" orientation="landscape">
<page height="396" width="612">
```

<IMAGE>

Places an image on the document.

All attributes, except for "`FILE`", are optional.

FILE Specifies the filename. (No default. Filename must be specified.)

ROTATE Rotate the image. (Default: 0.0 – no rotation.)

HEIGHT Specifies the height. (Default: image's actual height.)

WIDTH Specifies the width. (Default: image's actual width.)

X X position. (Default: current X position.)

Y Y position. (Default: current Y position.)

There is no way to explicitly set the Z-order of images. The image will be "above" any text that comes earlier on the page, and "under" any text that comes later on the page.

If no explicit path is given to the filename, filePro will search the following, in this order:

- The main filePro file's default directory.
- PFIMAGEDIR
- PFDLDIR

The filename is case-sensitive if the underlying O/S usescase-sensitive filenames. (ie: Unix/Linux are case-sensitive, while Windows is not.)

Currently, only PNG (*.png) and JPEG (*.jpg or *.jpeg) image files are supported.

The rotation is specified in degrees, counter-clockwise.

Height and width are specified in points.

If you specify "scale", then that direction will scale proportionately, based on the size given the other direction. For example, if the image has a height of 75, then specifying:

HEIGHT="150" WIDTH="scale"

will double the width as well. Specifying one direction and not the other will leave the other unchanged.

The position on the page (default: the current position) is done with the X/Y attributes. These can be a number, specified in points, or one of "LEFT"/"CENTER"/"RIGHT" for "X", and "TOP"/"CENTER"/"BOTTOM" for "Y".

Note that there is currently no resampling of the image. The image is printed at full resolution, just scaled to the specified size.

Example:

```
<image file="letterhead.png" x="top" y="center">
<image file="watermark.jpg" x="center" y="center" rotate="45">
```

<FONT>

Sets the font as specified.

All attributes are optional.

Note that font names are case-significant.

NAME The name of the font.

PDF supports 14 "base" fonts:

"Courier" (plain, bold, italic, bold-italic)

"Helvetica" (plain, bold, italic, bold-italic)

"Times" (plain, bold, italic, bold-italic)

"Symbol" (plain only)

"ZapfDingbats" (plain only)

Other fonts can be loaded via the PFFONT_* environment variables. (See below.)

SIZE Size of the font, in points.

BOLD Turns bold ON or OFF.

ITALIC Turns italics ON of OFF.

UNDERLINE Turns underlining ON or OFF.

ENCODE Specifies the character set encoding for characters 128-255.

Possible values are:

StandardEncoding	It is the default encoding of PDF
MacRomanEncoding	The standard encoding of Mac OS
WinAnsiEncoding	The standard encoding of Windows
FontSpecific	Use the built-in encoding of a font.
ISO8859-2	Latin Alphabet No.2
ISO8859-3	Latin Alphabet No.3
ISO8859-4	Latin Alphabet No.4
ISO8859-5	Latin Cyrillic Alphabet
ISO8859-6	Latin Arabic Alphabet
ISO8859-7	Latin Greek Alphabet
ISO8859-8	Latin Hebrew Alphabet
ISO8859-9	Latin Alphabet No. 5
ISO8859-10	Latin Alphabet No. 6
ISO8859-11	Thai, TIS 620-2569 character set
ISO8859-13	Latin Alphabet No. 7
ISO8859-14	Latin Alphabet No. 8
ISO8859-15	Latin Alphabet No. 9
ISO8859-16	Latin Alphabet No. 10
CP1250	Microsoft Windows Codepage 1250 (EE)
CP1251	Microsoft Windows Codepage 1251 (Cyrl)
CP1252	Microsoft Windows Codepage 1252 (ANSI)
CP1253	Microsoft Windows Codepage 1253 (Greek)
CP1254	Microsoft Windows Codepage 1254 (Turk)

CP1255	Microsoft Windows Codepage 1255 (Hebr)
CP1256	Microsoft Windows Codepage 1256 (Arab)
CP1257	Microsoft Windows Codepage 1257 (BaltRim)
CP1258	Microsoft Windows Codepage 1258 (Viet)
KOI8-R	Russian Net Character Set

COLOR Color

Currently, the only way to specify colors is "#rrggbb", where "rr", "gg", and "bb" are the two-digit hex value for red, green, and blue, respectively.

The default is Courier 10, black, with "standard" encoding. If the name, size, color, or encoding is not specified, that attribute remains unchanged.

Note that specifying a font name will clear bold/italic/underline, unless explicitly set to "on" in the same code.

Note that PDF does not support underlined text. Instead, this is accomplished by drawing a line underneath the text.

Example:

```
<font name="Helvetica" size="14" bold="on">
```

<MOVETO>

Move the current text position as specified.

All attributes are optional.

X Specifies the X position.

Y Specifies the Y position.

If a coordinate is not specified, the position on that axis remains unchanged.

Units can be absolute or relative. (If the value starts with "+" or "-", it is relative to the current position.)

Example:

```
<moveto y="-6">raised text<moveto y="+6">
```

<RECT>

Draw a rectangle.

All attributes are optional.

X Specifies the X position of one corner.

Y Specifies the Y position of one corner.

WIDTH The width of the rectangle.

HEIGHT The height of the rectangle.

STROKE The width of the line.

COLOR Color of the line.

Draw a rectangle. If (X,Y) is not specified, the current position is used.

Note that a height of 0 draws a horizontal line, and a width of 0 draws a vertical line.

```
<rect height="72" width="360" stroke="3.5">
```

PFFONT_*

Although PDF only includes the built-in "base14" fonts listed above, additional fonts can be used by defining them with the "PFFONT_*" environment variable. The format is:

```
PFFONT_name=embed:base:bold:italic:bold-italic:
```

(Note: On Windows platforms, the semicolon is used rather than the colon as the separator.)

Where:

"name" is the name by which you will refer to the font. For example, setting the environment variable "PFFONT_Fancy" would create a font named "Fancy", which could be referenced in any <FONT> tag. (Remember that font names are case-sensitive.)

"embed" specifies whether the font is to be embedded in the PDF document. If a font is not embedded, the document may not be displayed properly if the system on which it is viewed does not have that font installed. However, embedding the font makes the PDF document larger, and copyright restrictions may prevent a font from being embedded. The allowed values are "0" (do not embed) and "1" (embed).

"base" specifies the filename which contains the base font description, and is required.

"bold", "italic", and "bold-italic" specify the filenames containing the font description for bold, italic, and bold-italic, respectively. These are all optional, and if left off will cause any reference to the font to revert to the "base" font.

If a path is not specified in the filename, filePro searches for the file in the following directories:

- PFFONTDIR
- PFDLDIR
- (Windows only) The Windows system font directory.

Currently, only TrueType (*.ttf) fonts are supported. The filename extension must be included.

Note that the built-in base14 fonts can be overridden using the PFFONT_* variable. For example, setting PFFONT_Courier=1:cour.ttf:courbd.ttf:couri.ttf:courbi.ttf would replace the built-in "Courier" font with the specified TrueType font.

FPML output

If you set the new "allow embedded FPML on form" option in dmoedef, you can include FPML codes directly on the output format.

You can also generate "on-the-fly" FPML by putting the FPML in a field on the form, prefaced with ESC. For example:

```
If:
```

```
  Then: color = "#000000"  ' black
    If: value lt "0"
  Then: color = "#800000"  ' red
    If:
  Then: xx = chr("27") & "<FONT COLOR='" { color { "'>" { value
            { chr("27") & "<FONT COLOR='#000000'>"
```

Finally, note that you can put FPML directly on the form, and still fill in values at runtime, by putting something like this on the form:

```
<FONT COLOR='*aa      '>*bb        <FONT COLOR='#000000'>
```