Hans Hagen
Ridderstraat 27
8061GH Hasselt
pragma@wxs.nl

pdftex

pdfTEX's little secret
Tracking positions

keywords
pdf, pdfTEX, ConTEXt, emTEX, specials, positioning

abstract
It is no secret that pdfTEX introduces new primitives, but some of them are less known than others. In this article I will describe an example of the use of \pdfsavepos cum suis. We will investigate their usage by implementing a simple emTEX specials simulator.

When Hàn Thế Thành, the author of PDFTEX, and I were exchanging some emails on PDFTEX functionality, positional information popped up as potential extension. Actually, it did not take that much time to cook up the basic functionality and the author had implemented it before I could even start to think about real advanced applications.

I'm sure that TEX programmers can spend many days on how and what kind of information is needed if you want to have access to positions, but since high level macros will probably be used anyway, even things like multiple reference points have proved to be rather unimportant at the system level.

Therefore, PDFTEX provides just these three primitives:

`\pdfsavepos`	marks the current position
`\pdflastxpos`	the last marked horizontal position
`\pdflastypos`	the last marked vertical position

Based on these three primitives, very advanced systems can be build, and for some time now, CONTEXT has such a system in its core. However, not everyone uses CONTEXT, so we will demonstrate position tracking in generic applications.

Because PDFTEX produces its output directly, many of those nice tricks provided by back–ends by means of `\special` fail when producing PDF code directly. Take for instance EMTEX specials. When someone sent me a mail asking if PDFTEX did support those specials, the original answer was “no”, but in the last few years I have learned that you must never underestimate TEX's capabilities.

I must admit that I never use those specials myself, but from the way they were used in the macros I was sent, I learned that they depend on the back–end's capability to access the current position. For those who know TEX this may be bad news, since pure TEX does not provide any positional information. So in order to use those specials, you must be sure that they are supported by every driver you use. However, the good news is that PDFTEX does support position tracking, so here is our generic example.

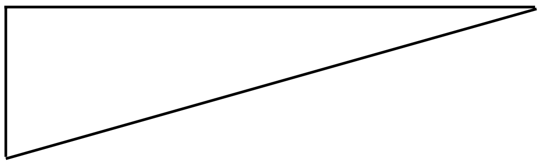
The two EMTEX specials we need to implement are packaged into the macros `\EMmoveto` and `\EMlineto`, like:

```
\def\EMmoveto{\special{em:moveto}}
\def\EMlineto{\special{em:lineto}}
```

They are used in macro packages to draw lines, and the results are often boxes with content like the following:

```
\vbox to 2cm
  {\offinterlineskip \EMmoveto
   \hskip 7cm        \EMlineto
   \vskip 2cm        \EMlineto
   \vskip-2cm        \EMlineto}
```

This box will contain a triangle, and when typeset, it should look like:



These two macros can be implemented as follows. When moving to a position, we only have to register the new coordinates. Once they are known, we use them to draw a line and afterwards we save these end coordinates as starting point for the next line segment. So, at each point specified by `\EMlineto` we need to know the coordinates.

```
\def\EMmoveto%
  {\EMgetposition\EMlastmovex\EMlastmovey}
```

The primitives `\pdflastxpos` and `\pdflastypos` return a number, representing the x and y coordinate in scaled points, TEX's smallest unit of length. We need to convert this number into base points as used by PostScript and PDF. When done, we insert some literal PDF code into the text using `\pdfliteral`. Here, the `m` means 'moveto', the `l` means 'lineto' and the `S` operator 'strokes' (draws) the line.

```
\def\EMlineto%
  {\bgroup
   \EMgetposition\EMlastlinex\EMlastliney
   \count0=\EMlastmovex \advance\count0 by -\EMlastlinex
   \count2=\EMlastmovey \advance\count2 by -\EMlastliney
   \divide\count0 by 65536
   \divide\count2 by 65536
   \pdfliteral{1 w 0 0 m \the\count0 \space\the\count2 \space l S}%
   \global\let\EMlastmovex\EMlastlinex
   \global\let\EMlastmovey\EMlastliney
   \egroup}
```

We need a fresh start, so we first set the current position to zero.

```
\def\resetEMspecials%
  {\gdef\EMlastmovex{0}\gdef\EMlastmovey{0}}
```

Next comes the macro that keeps track of the position. The current position is marked with `\pdfsavepos` and its coordinates are written to a file whenever the page is shipped out, since `\write` postpones its action until that moment. The file has entries like:

```
\EMsetpos 1 4661756 46651918
\EMsetpos 2 5000359 46990521
\EMsetpos 3 4661756 46313315
\EMsetpos 4 5338962 46990521
\EMsetpos 5 4661756 45974712
```

These lines are written with the command:

```
\write\EMfile
  {\EMsetpos\number\EMcounter
   \space\number\pdflastxpos\space\number\pdflastypos}%
```

In reality the argument to `\write` looks slightly more complicated, because we have to make sure that the number of the current position is frozen and `\EMsetpos` is not expanded. We do so by explicitly expanding the number beforehand and preventing expansion of `\EMsetpos`.

```
\def\EMgetposition#1#2%
  {\bgroup
   \pdfsavepos
   \global\advance\EMcounter by 1
   \expandafter\write\expandafter\EMfile\expandafter
     {\expandafter\noexpand\expandafter\EMsetpos\number\EMcounter
      \space\number\pdflastxpos\space\number\pdflastypos}%
   \EMsetcounters
   \xdef#1{\the\count0}%
   \xdef#2{\the\count2}%
   \egroup}
```

The counter mentioned a few lines ago needs to be declared before it can be used.

```
\newcount\EMcounter
```

We also need a dedicated file slot.

```
\newwrite\EMfile
```

Before we open the file for writing, we read in the data written in the previous pass, but only if the file is present.

```
\def\EMfilename{\jobname.emp}

\def\startEMspecials%
  {\resetEMspecials
   \openin\scratchread=\EMfilename \relax
   \ifeof\scratchread\else \input \EMfilename \relax \fi
   \closein\scratchread
   \immediate\openout\EMfile=\EMfilename\relax}

\def\stopEMspecials%
  {\closeout\EMfile}
```

Just to be sure, we test if `\scratchread` is defined, and if not, we allocate a slot.

```
\ifx\undefined\scratchread \newread\scratchread \fi
```

This leaves us two commands. The `\EMsetpos` command that ends up in the file stores each position in a macro. When this macro is expanded, it assigns the coordinates to two scratch counters.

```
\def\EMsetpos#1 #2 #3 % number x y
  {\expandafter\xdef\csname EM:#1\endcsname{\count0=#2 \count2=#3}}
```

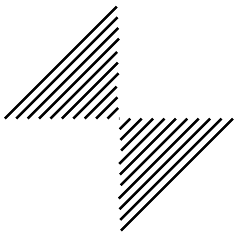
This position is recalled with its companion macro. First we set the counters to zero. When the position is unknown, nothing happens since the `\csname...` will expand to `\relax`.

```
\def\EMsetcounters%
  {\count0=0 \count2=0
   \csname EM:\the\EMcounter\endcsname}
```

These macros are rather independent of the macro package you use. For instance, in ConTEXt the following works well:

```
\setuppositioning[unit=ex]
\startpositioning
  \dostepwiserecurse{-10}{10}{1}
    {\position(0,\recurselevel){\EMmoveto}
     \position(\recurselevel,0){\EMlineto}}
\stoppositioning
```

Here, we hook the EMTEX macros into an existing text positioning mechanism, which positions the commands using TEX's skips and kerns.



Of course one should start and end the file with:

```
\startEMspecials
\stopEMspecials
```

and, if needed, reset the begin position at each page using:

```
\resetEMspecials
```

A few pages ago, we mentioned that ConTEXt has built–in position tracking. This means that when we want to implement this kind of trickery in this macro package, we can fall back on existing functionality. In the following alternative we will also use a few skips. This keeps the source readable and ConTEXt has plenty of unused registers to accomodate this strategy.

```
\newcount\EMcounter \def\EMvariable{EM:\the\EMcounter}

\newskip \EMlastmovex \newskip \EMlastmovey
\newskip \EMlastlinex \newskip \EMlastliney

\def\resetEMspecials%
  {\global\EMlastmovex=0pt \global\EMlastmovey=\EMlastmovex}

\resetEMspecials \appendtoks\resetEMspecials\to\everyshipout
```

Watch how we reset the specials after a page is flushed. We don't have to bother about files here, because saving and recalling is already implemented. Although not needed, we define the start–stop macros, so that ConTEXt users who key them in are not confronted with error messages.

```
\let\startEMspecials\relax
\let\stopEMspecials \relax

\def\EMgetposition#1#2%
  {\global\advance\EMcounter 1
   \setposition\EMvariable
   \global#1=\POSx\EMvariable
   \global#2=\POSy\EMvariable}
```

```
\def\EMmoveto%
  {\EMgetposition\EMlastmovex\EMlastmovey}

\def\EMlineto%
  {\EMgetposition\EMlastlinex\EMlastliney
   \global\advance\EMlastmovex -\EMlastlinex
   \global\advance\EMlastmovey -\EMlastliney
   \ScaledPointsToBigPoints{\number\EMlastmovex}\EMx
   \ScaledPointsToBigPoints{\number\EMlastmovey}\EMy
   \pdfliteral{1 w 0 0 m \EMx \space \EMy \space l S}%
   \global\EMlastmovex\EMlastlinex
   \global\EMlastmovey\EMlastliney}
```

The command `\setposition` registers a position by name (here `\EMvariable`), while `\POSx` and `\POSy` give you access to the coordinates.

These three commands are contained in a suite of low level commands that can be used to register and get access to positional information. The current mechanism is not yet complete, but already provides enough hooks for advanced embedded graphics. Its functionality is a natural extension to the METAPOST support already present in ConTEXt. Therefore, more advanced examples can be found in the MetaFun manual, since they fall beyond the scope of this introductory article.

As a bonus, I will now provide a few macros that will make this mechanism transparant to DVI as well as PDF output. We will use `\pdfoutput` as trigger.

```
\ifx\pdfoutput\undefined \chardef\pdfoutput=0 \fi
```

We save some of the macros we defined previously:

```
\let\pdfEMmoveto\EMmoveto
\let\pdfEMlineto\EMlineto

\let\pdfstartEMspecials\startEMspecials
\let\pdfstopEMspecials \stopEMspecials
```

We now redefine them to support DVI and PDF.

```
\def\EMmoveto%
  {\ifcase\pdfoutput\special{em:moveto}\else\pdfEMmoveto\fi}
\def\EMlineto%
  {\ifcase\pdfoutput\special{em:lineto}\else\pdfEMlineto\fi}

\def\startEMspecials%
  {\ifcase\pdfoutput\else\pdfstartEMspecials\fi}
\def\stopEMspecials%
  {\ifcase\pdfoutput\else\pdfstopEMspecials \fi}
```

If there is any real demand for this in ConTEXt, I will hook these macros in the special drivers, so that their support becomes more natural.

You may wonder to what extent positional tracking is PDFTEX specific. In ConTEXt, we also support position tracking in DVI by using specials and analyzing the DVI file afterwards using a Perl script created by Taco Hoekwater. Since many of the advanced TEX features depend on some kind of back–end, we don't consider it to be a disadvantage. Of course, the PDFTEX way is not only cleaner, but also faster. It was more out of curiosity than out of need that we provided the DVI methods as well. Also, it is always good to have more roads to reach the same goal.