AcroTeX.Net

GraphicxSP

Using and re-using Embedded Graphics

D. P. Story

Prepared: February 19, 2016

www.acrotex.net
Version 1.0

Table of Contents

1. Introduction

Beginning with Version 5.0, "Acrobat Distiller allows a PostScript language program to specify that a given set of graphical operations should be encapsulated and treated as a single object. The pdfmarks **BP** (Begin Picture) and **EP** (End Picture) enclose a set of graphic operations. The **SP** (Show Picture) pdfmark indicates where to insert an object (which may be inserted in more than one place)." [1] This is the approach taken by this package.

GraphicxSP includes commands for embedding an EPS graphic (using the **BP** and **EP** pdfmark operators) in a PDF derived from a LaTeX source and for displaying the graphic, possibly multiple times. For documents that use the same graphic multiple times (for backgrounds, watermarks, company logos, appearances for form fields, etc.) using GraphicxSP will (greatly) reduce the size of the PDF.[2]

The PDF 1.4 (Acrobat version 5.0) introduces the notion of transparency into the Adobe imaging model. Support for transparency through the pdfmark paradigm does not appear until Version 6.0. [3] This package also supports transparency feature through the **SetTransparency** pdfmark.

GraphicxSP, as the name might suggest, is an extension of the graphicx package. Functionality for GraphicxSP is accessed through the `\includegrapics` command. Additional key-values for the `\includegrapics` command are defined that support the use GraphicxSP.

2. Requirements

LaTeX, PDF, and transparency requirements are discussed in this section.

2.1. LaTeX package requirements

This package requires the graphicx package (by D. P. Carlisle and S. P. Q. Rahtz), the eso-pic package (by Rolf Niepraschk) and the everyshi package (by Martin Schröder). GraphicxSP redefines a few commands of the first two packages: the graphics/graphicx package commands `\Gin@setfile` and `\Gin@getbase` are redefined, as is the eso-pic command `\@ShipoutPicture`.

AeB (AcroTeX eDucation Bundle) is not a required package, but to successfully compile the example `grxsp_forms_aeb.tex`, you need the latest AeB as some modification to the eforms package were made. See http://www.math.uakron.edu/~dpstory/graphicxsp.html for the latest version.

2.2. PDF creator requirements

The big restriction on this package is the requirement to use **Acrobat Distiller** (version 5.0, or version 6.0 for transparency). The package was developed using Acrobat Distiller 8.1. The package supports the creation of PostScript using dvips and dvipsone.

[1] Section 4.6.1 of the *pdfmark Reference Manual*, Technical Note #5150, Version 5.0

[2] The amount of reduction in file size depends on how many times a particular graphic is repeatedly used.

[3] Section 4.6.1 of the *pdfmark Reference Manual*, Technical Note #5150, Version 6.0, page 33.

Though distiller is required, there is no reason why, however, this package can't be modified to work for any PDF creator that supports the pdfmark; in particular, I invite anyone to extend this package to GhostScript.

2.3. Transparency requirements

To get the transparency effect, Acrobat Distiller version 6.0 or later is required. The default setting of the distiller does not support the **SetTransparency** pdfmark; it is necessary to edit the `.joboptions` file.

The procedure for editing `.joboptions` to support transparency is as follows:

1. Start Acrobat Distiller
2. From the Default Settings list, select the setting you want to edit, usually, this will be the `Standard` job options.
3. Select Settings > Edit Adobe PDF Settings (Ctrl+E) from the distiller menu.
4. Click the `SaveAs` button at the bottom of the `Adobe PDF Settings` dialog box. Save your `.jobsettings` file under a new name, say `Standard_transparency` and make a note of where the distiller saves this file.
5. With your favorite text editor, navigate to the folder where you saved your new `.joboptions` file, and open it in your editor.
6. Look for the line `/AllowTransparency false`, change `false` to `true` so that it now reads `/AllowTransparency true`. Save the changes and close the file.
7. Use this `.joboptions` file, `Standard_transparency` for example, whenever you distill with transparency pdfmarks. If your LaTeX file uses transparency, and you are using a `.joboptions` file with `/AllowTransparency false`, distillation will fail and the distiller log should say

```
%%[Error: The PostScript contains Transparency pdfmark, job aborted.]%%
%%[ /AllowTransparency is false in job option settings.]%%
%%[ Error: undefined; OffendingCommand: pdfmark;
    ErrorInfo: Transparency Group ]%%
```

   This suggests that you should use your `.joboptions` file that supports transparency!

3. Options of this package

Currently, there is a minimal of package options

- `dvips`: Use this option if you are a user of most any other TeX systems available to the community.
- `dvipsone`: Use this option if you are a user of the Y&Y TeX system to create your PostScript file.

- `preview`: The dvi previewers are designed to show a preview of the graphic, these previewers know nothing about the **SP** pdfmark (Show Picture). If you use the `preview` option, a `\fbox` is placed around all graphics inserted by the GraphicxSP package. This option sets a boolean switch, `\ifpreview` to true. You are free to locally set this switch to get the `\fbox` to appear, use include `\previewtrue` in your document, or, to turn the preview off again, type `\previewfalse`.
- `showembeds`: As just mentioned, dvi previewers are designed to preview graphics; consequently, the graphic files embedded by GraphicxSP, will have a preview as well. The embedding command `\embedEPS` is required to appear only in the preamble, so the preview for this embedded graphics always appears on the first page. GraphicxSP covers these embedded files using a white `colorbox`; content and other graphics are placed on top of the white `colorbox`. Using the `showembeds` will cause this white color box not to be created, hence, you can see the dvi preview of the embedded graphics, stacked one on top of the other.
- `draft`: Similar to the option of the same name in the graphicx package. Instead of displaying an image, a rectangular box is displayed with the name of the graphic in its interior.
- `shownonames`: If this option is chosen along with the `draft` option, the name of the graphic is not displayed.

4. The GraphicxSP commands

In this section we present the new commands defined in the GraphicxSP package.

4.1. Embedding commands

Before we can show a graphic, we first must embed it using the `\embedEPS` command.

`\embedEPS[⟨key-values⟩]{⟨name⟩}{⟨path⟩}`

Command Location: This command is restricted to the preamble.

Parameter Description: The parameters are as follows:

1. The first optional parameter takes key-value pairs:
   - `hiresbb`: This is a key from the graphicx package. When this key is present, graphicx will look for the high resolution bounding box; otherwise, is looks for the bounding box.
   - `transparencyGroup`: Use this key if the embedded graphic is to be used with transparency. The key is normally used by itself, but it can be given values, a good old-fashion reading of Chapter 7 on Transparency in the PDF Reference, sixth edition, version 1.7, is necessary.
2. The second parameter ⟨*name*⟩ is the symbolic name for the graphic. This symbolic name must be unique to the document (or distiller will crash); GraphicxSP will stop compilation if two embedded EPS files are given the same name.

3. The third parameter is the path to the graphic, an EPS graphic. The graphic can be in the current folder, or on the graphics search path. The same rules hold here as in the graphicx package.

For example,

```
\embedEPS[hiresbb,transparencyGroup]{myBestPic}{graphics/AdobeDon}
```

When a file is embedded, certain parameters are saved and are available for use by the document author. Each of the commands below take the symbolic name, ⟨*name*⟩ of the graphic as their only argument.

- `\bboxOf{`⟨*name*⟩`}`: The dimensions of the bounding box, `llx lly urx ury`, of the graphic with symbolic name ⟨*name*⟩. The individual values of this bounding box can be accessed by the next four commands.
- `\llxOf{`⟨*name*⟩`}`: The lower left x-coordinate of the (high res) bounding box as read by graphicx.
- `\llyOf{`⟨*name*⟩`}`: The lower left y-coordinate of the (high res) bounding box as read by graphicx.
- `\urxOf{`⟨*name*⟩`}`: The upper left x-coordinate of the (high res) bounding box as read by graphicx.
- `\uryOf{`⟨*name*⟩`}`: The upper left y-coordinate of the (high res) bounding box as read by graphicx.
- `\heightOf{`⟨*name*⟩`}`: The height of the graphic, based on the bounding box information, `\uryOf{`⟨*name*⟩`}` - `\llyOf{`⟨*name*⟩`}`.
- `\widthOf{`⟨*name*⟩`}`: The width of the graphic, based on the bounding box information, `\urxOf{`⟨*name*⟩`}` - `\llxOf{`⟨*name*⟩`}`.

Each expands to a number representing Adobe points. To convert these numbers for use by TEX just add the suffix `bp`, e.g., in TEX space, the height of the graphic is `\heightOf{`⟨*name*⟩`}bp`.

These commands are useful when creating variations on an `\embedEPS` graphic using the `createImage` environment defined below, or for creating form fields with an `\embedEPS` graphic as an appearance.

The `\embedEPS` command is designed for graphics that are in an exterior EPS file. You can also define a graphic, or image, using raw PostScript using the `createImage` environment.

```
\begin{createImage}[⟨key-values⟩]{⟨bbox⟩}{⟨name⟩}
    ⟨postscript code⟩
\end{createImage}
```

Parameter Description: The parameters are as follows:

1. The first optional parameter takes key-value pairs:
   - `transparencyGroup`: Use this key if the embedded graphic is to be use with transparency. The key is normally used by itself, but it can be given values, a good old-fashion reading of Chapter 7 on Transparency in the PDF Reference, sixth edition, version 1.7, is necessary.
2. The second parameter, ⟨*bbox*⟩, is the bounding box of the image being created.
3. The third parameter ⟨*name*⟩ is the symbolic name for the graphic. This symbolic name must be unique to the document (or distiller will crash); GraphicxSP will stop compilation if two embedded EPS files are given the same name.

The following image appears in one of the demo files of this package, and is code written many years ago for a tic-tac-toe game in PDF.

```
\begin{createImage}{0 0 100 100}{xO}
.7529 setgray 0 0 100 100 rectfill 1 setgray 2 2 moveto 2 98 lineto
98 98 lineto 96 96 lineto 4 96 lineto 4 4 lineto fill 0.34 setgray
98 98 moveto 98 2 lineto 2 2 lineto 4 4 lineto 96 4 lineto
96 96 lineto fill 0 setgray 22.5 22.5 moveto 1 0 0 setrgbcolor
/Helvetica 72 selectfont (O) show
\end{createImage}

\embedEPS[hiresbb,transparencyGroup]{myBestPic}{graphics/AdobeDon}
```

we can then manipulate this image a little

```
\begin{createImage}{\bbox{myBestPic}}{upsideDownAD}
     0 \heightOf{myBestPic} rmoveto
     currentpoint translate
     1 -1 scale [ {myBestPic} /SP pdfmark
\end{createImage}
```

Notice that we reposition the graphic to fit inside the bounding box.

I don't mean to get ahead of myself, but you can introduce transparency as well.

```
\begin{createImage}{\bbox{myBestPic}}{transparentAD}
    [ /ca .3 /SetTransparency pdfmark
    [ {myBestPic} /SP pdfmark
\end{createImage}
```

All these examples will appear again below when `\includegraphics` and `\insertEPS` are discussed.

The commands `\llxOf`, `\llyOf`, `\urxOf`, `\uryOf`, `\bboxOf`, `\heightOf` and `\widthOf` are also defined for images created by the `createImage` environment.

4.2. Inserting graphics commands

Once we have embedded our graphics using `\embedEPS` or create a new image or modified an old image using the `createImage` environment, we can display them to our heart's content. There are two commands for displaying a embedded graphic, `\includegraphics` and `\insertEPS`.

```
\includegraphics[name=⟨name⟩,⟨key-values⟩]{⟨path⟩}
\insertEPS[⟨key-values⟩]{⟨name⟩}
```

Parameter Description: The ⟨*name*⟩ key is required in the `\includegraphics` command, the value is the symbolic ⟨*name*⟩ of the graphic to be used. When the name key is used, the ⟨*path*⟩ parameter is not used in GraphicxSP and can actually be empty; if the name key is not specified, then `\includegraphics` behaves just like it always does, includes the file in the document and displays it. Because ⟨*path*⟩ is not used with SP graphics, GraphicxSP offers `\insertEPS` as an alternative; here the ⟨*name*⟩ is given as the second argument. The ⟨*key-values*⟩ are described next.

Key-Value Pairs: Both commands take all the key-values of the `\includegraphics` command, plus a few more:

1. `name=`⟨*name*⟩: The symbolic name (⟨*name*⟩) for the graphic, it is required with the `\includegraphics` command, and is ignored in the `\insertEPS` command. In the later command, ⟨*name*⟩ is entered as the second argument.

2. `transparency=`⟨*key-values*⟩: Here you can enter various key-value pairs for setting transparency. These keys-values are listed in the pdfmark Reference in the section on the SetTransparency pdfmark. Of the ones listed, I've only explored **BM**, **ca** and **CA**. The following come from the pdfmark Reference on these three:

   - **CA**: Current stroking alpha constant, specifying the constant shape or constant opacity value to be used for stroking operations. Default is 1.0.
   - **ca**: Same as CA, but for nonstroking operation. Default is 1.0.
   - **BM**: Current blend mode. Default is Normal. Other blend modes are Normal, Multiply, Screen, Darken, Lighten, ColorDodge, ColorBurn, HardLight, SoftLight, Difference, Exclusion. See the PDF Reference for a description of these.

   The most useful of these is **ca**, and it is illustrated in the demo files and well as here.

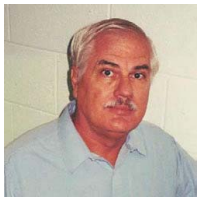
   Example: `transparency={/ca .3 /BM/Normal}`

3. `presp=`⟨*postscript*⟩: This allows you to insert PostScript commands just before **SP** pdfmark.

4. `postsp=`⟨*postscript*⟩: This allows you to insert PostScript commands just after **SP** pdfmark.

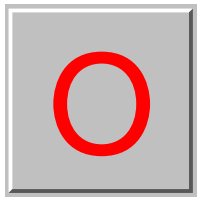
The latter two key-values can be used to create special effects, as illustrated below.

4.3. Examples



```
\includegraphics[name=myBestPic,width=1in]{}
```

Below is the image created not from a EPS file, but from PostScript language code.



```
\includegraphics[name=x0,width=1in]{}
```

The ⟨*path*⟩ argument can remain empty as it is not used when the name key is present. Now, we'll switch over to the shorter `\embedEPS` command.

Here is the image created earlier using the `createImage` environment. The same effect can be done with `\scalebox`.

```
\insertEPS[width=1in]{upsideDownAD}
```

Below is the transparent image created using the `createImage` environment. The same effect can be done by setting the `transparency` key set to `{/ca .3}`. The background is white and it is this white background you see showing through the image.

```
\insertEPS[width=1in]{transparentAD}
```

Next, we set the `transparency` key to an opacity value ranging from 70% on the left (`/ca .7`) to 30% on the right (`/ca .3`), and place a colorful background under the image so you can see the transparency of the image.

```
transparency={/ca .7}...transparency={/ca .3}
```

We can rotate the image using the usual graphicx controls, as below, or we can use `\rotatebox`.

```
\insertEPS[width=1in,angle=45]{myBestPic}
```

Any of the keys of `\includegraphics` can be used to manipulate the image. The images can also be manipulated using other graphicx commands, `\resizebox`, `\scalebox` and `\rotatebox`, for example.

Finally, here is an example of the use of `presp` and `postsp`:

The `presp` code is

```
\def\mypreSP#1{%
    newpath
    \widthOf{#1} 2 div \heightOf{#1} 2 div
    \widthOf{#1} 2 div \heightOf{#1} 2 div
    Draw_Ellipse
    clip
    newpath
}
```

and clips the image in the shape of an ellipse. In this case, the image is almost square, so the image is almost circular.

The postsp is

```
\def\mypostSP#1{%
    gsave
    [ /ca .4 /SetTransparency pdfmark
    \widthOf{#1} 2 div \heightOf{#1} 2 div
    \widthOf{#1} 2 div \heightOf{#1} 2 div
    Draw_Ellipse
    0.4 0.7 1 setrgbcolor
    fill
    grestore
    gsave
    [ /CA .5 /BM/Screen /SetTransparency pdfmark
    \widthOf{#1} 2 div \heightOf{#1} 2 div
    \widthOf{#1} 2 div \heightOf{#1} 2 div
    Draw_Ellipse
    40 setlinewidth
    0.4 0.7 1 setrgbcolor
    stroke
    grestore
}
```

After clipping the image itself, and displaying the image, we then cover the image with an ellipse the same size, set the transparency to 40% opacity and fill, giving the image a bluish haze. Next, draw the ellipse again and stroke it with a line 40 points wide. Give it an opacity of 50%, so AdobeDon will shine through.

5. Special techniques for form appearances

In addition to using embedded graphics to display an image with `\includegraphics` or `\insertEPS`, these images can be used as appearances for form fields. Examples of a push button and checkbox can be found in the demo files, here reproduce the push button example.

For this techniques, the eforms package is required. The forum support of hyperref may not support the **AP** key the way eforms does.

In the preamble, we have

```
\begin{createImage}{\bboxOf{myBestPic}}{nAdobeDon}
    [ {myBestPic} /SP pdfmark
\end{createImage}

\begin{createImage}{\bboxOf{myBestPic}}{rAdobeDon}
    [ /ca .5 /SetTransparency pdfmark
    [ {myBestPic} /SP pdfmark
\end{createImage}
```

```
\begin{createImage}{\bboxOf{myBestPic}}{dAdobeDon}
    [ /ca .3 /SetTransparency pdfmark
    [ {myBestPic} /SP pdfmark
\end{createImage}
```

We use `\bboxOf` to set the bounding boxes to be the same as the original `myBestPic` image, and define a normal, rollover and down appearances for the button (`nAdobeDon`, `rAdobeDon` and `dAdobeDon`), respectively. We use various opacity settings for the appearances.

The button code is

```
\resizebox{.5in}{!}{\pushButton[%
    \autoCenter{n}
    \A{\JS{app.alert("AcroTeX rocks the world!"); this.dirty=false;}}
    \AP{/N {nAdobeDon} /D {dAdobeDon} /R {rAdobeDon}}
]{pbAdobeDon}{\widthOf{myBestPic}bp}{\heightOf{myBestPic}bp}}
```

See the eforms manual for details of these key-value pairs.

Finally, the button looks like this: 

When using **SP** images for form fields in this way, the images **must always pass** through a `createImage` environment where the bounding boxes can be precisely set, in the verbatim listing above, note `\bboxOf{myBestPic}` sets the bounding box. The reason you can't use an embedded image directly is that the embedded image has had its bounding box reset to an enormous value. See the documentation in graphicxsp.dtx for more details.

The dimensions of the push button are the last two arguments of the `\pushButton` command and are set to `\widthOf{myBestPic}bp` and `\heightOf{myBestPic}bp`, (Note the use of the `bp` dimension.) We then resize this field using `\resizebox`! Very swave how the graphicx and GraphicxSP work together.

6. Tips

Since you are using distiller, you have Acrobat as well. Try using the PDF Optimizer (`Advanced > PDF Optimizer` menu) to further reduce the size of the file. If you have Acrobat Pro 8.0, you can do a Save As, by selecting Adobe PDF Files, Optimized from the Save as type list. This is the same as using the PDF Optimizer.

An example of the savings is this manual, after distillation document size was 176.6 KB, after the PDF Optimizer was done, the file size was 113.6 KB. That's a reduction of 63 KB or that's a reduction of 38%. For some of the demo files, the reduction is much more dramatic since the images are used and re-used many more times than in this document.

7. GraphicxSP example files

The example files can be found in the `examples` folder of the GraphicxSP distribution. They are

- `grxsp_tst_noaeb.tex`: General test file demonstrating SP graphics, does not require AeB (AcroTEX eDucation Bundle).
- `grxsp_tst_aeb.tex`: Same as previous file, but using AeB.
- `grxsp_comp_noaeb.tex`: A comparison between graphicx `\includegraphics` and SP graphics. AeB not used.
- `grxsp_comp_aeb.tex`: Same as previous file, but AeB is used.
- `grxsp_forms_aeb.tex`: This file demonstrates using SP graphics as appearances of form fields. AeB required.
- `grxsp_layers_aebpro.tex`: Demonstrates the use of SP graphics with Optional Content Groups, or layers. AeB Pro required.

That's all for now, I simply must get back to my retirement. ☺