# Fancy Imports of JPEG files to PostScript or Acrobat

**Don Lancaster**
**Synergetics, Box 809, Thatcher, AZ 85552**
**copyright c2003 as GuruGram #23**
**http://www.tinaja.com**
**don@tinaja.com**
**(928) 428-4073**

Most layout or pagemaking programs have one or more methods to import .JPEG photo images into **Acrobat** files that you are creating. Just about all better grade image conversion utilities can also handle this task. If all else fails, you can simply print the .JPEG photo to **Distiller** to get a .PDF page. And optionally print that page to disk if you want to get fancy with custom **PostScript** work.

But there's times and places where you would like to get at and understand the underlying **PostScript** code so you can do fancier custom stuff on your own terms. So, I'd like to briefly go over the JPEG conversion fundamentals here. And show you how to build an autopositioning, autosizing, autotracking, **and** autoURLing picture inserter. Yes, the hotbox automatically adjusts and repositions itself on any later text editing, guaranteeing you painless links.

We first looked at the fundamentals of .JPEG conversion back in **JPEG2PDF.HTML**. Where we saw that the two key secrets included making your .JPEG file locally available on hard disk and using the **/DCTDecode filter** input filter while creating your image. Additional details on this are in the **PostScript Reference Manual 3**.

Let's start with some autopositioning image code and then see what it will take to get it to work. While this makes use of my **gonzo utilities**, you should easily be able to adapt it to any host that gives you coding access…

```
/autofigure1 {
  save /af1 exch store              %  save graphics state
  xpos ypos translate               %  position on page
  6 0.65 923 526 .025               % xpos ypos xres yres scale
  (http://www.tinaja.com/images/bargs/flu84501.jpg)
  (C:\\windows\\desktop\\aaraw_pix\\flu84501.jpg)
  jpegimageprocwithlink             %  call conversion proc
  af1 restore                       %  And restore
               } store
```

In this case, **xpos** and **ypos** are **Gonzo** specific current positioning info. You can substitute hard numbers for these when doing an absolute positioning. The first two numbers on the next line are an **additional** x and y offset. These may be useful to fine tune a location if an image is wider or taller than usual.

The next two numbers are the .JPEG horizontal and vertical image resolution…

**JPEG size MUST be exact or the image will tear badly!**

If you do not already know your JPEG image size, load it into **ImageViewer32** or a similar utility, and read the size from the info line. The final value on the line is a **pixel scaling factor** that you can use to resize your JPEG image to the available space. Note that **Gonzo** often uses a 10X grid; your scaling may be larger.

This is followed by the url for the image clickthrough and then by the JPG image link itself. As usual, note this important rule…

**Any SINGLE reverse slash in a PostScript filename string MUST be replaced by a DOUBLE reverse slash!**

Our routine finally calls **jpegimageprocwithlink** that does all the work. Let's break this proc down into several pieces for analysis…

```
/jpegimageprocwithlink {
   save /snap2 exch def
   /infilename exch store       % grab passed pix file
   /inurllink exch store        % grab link filename
   /photoscale exch store       % grab scale
   /vpixels exch store          % grab v image size
   /hpixels exch store          % grab h image size

   translate                    % position on page
   inurllink setareaurl         % separate urllink proc
```

There are two separate goals to this code. We first want to create a tracking autopositioning link for image expansion or whatever. Followed by actual image capture and placement. Because much of the info needed by both routines is similar, it makes sense to nest them together. Otherwise, the same variables might have to be captured and passed twice.

Our proc starts with a save, then grabs all the needed values, does a translate to position itself on the page, and then calls **setareaurl** to create the actual link. Let's first look at this **setareaurllink** before continuing along with your main **jpegimageprocwithlink** code…

```
/setareaurl {
  /cururlname exch store         %   grab url
  mark                           % start pdfmark
  /Rect  [ 0 0                   % build hot zone
     hpixels photoscale mul
     vpixels photoscale mul ]
  /Border [ 0 0 0]               % no border
  /Color [ .7 0 0 ]              % but make it red anyhow

  /Action <</Subtype /URI /URI cururlname>>
  /Subtype /Link
  /ANN                           %  annotation type
  pdfmark                        %  call pdf operators
                  } def
```

This is pretty much straight out of the **PDFMark Reference Manual**. **PDF marks** are a way of including additional nonprinting instructions into a PDF file. This particular one creates a hot spot for a url clickthrough. The hot zone itself tracks our JPEG photo since it starts at the same 0,0 translation and is similarly scaled.

One additional tiny detail: Early in your PDF file, you'll want to make sure any PDF Marks are in fact non-printing so they do not present printer problems. Always insert this code early into most any PDF file you create…

```
/pdfmark where {pop}{userdict
    /pdfmark /cleartomark load put} ifelse
```

Now, let's continue on with our main **jpegimageprocwithlink** proc definition. The next order of business is to capture the incoming .JPEG file and convert it into an appropriately scaled PostScript image….

```
/DeviceRGB setcolorspace        %  pick color model
hpixels vpixels scale           %  magnify unit square
photoscale dup scale            %  rescale for area

/infile infilename (r) file def %  establish input read file
/Data {infile /DCTDecode filter} def   %  define data source
```

The image is first scaled from its 1x1 pixel space by its horizontal and vertical resolution. It is then rescaled to fit the available space. A readfile is defined that will grab data from the locally hard disk stashed .JPEG file.

Note that this .JPEG file is only needed during your Distill. All required image data is optionally resampled and then internalized and compressed for later .PDF distribution.

We continue by building an image dictionary, calling the image, restoring, and finally exiting...

```
      <<                            % start image dictionary
  /ImageType 1                      % always one
  /Width hpixels                    % JPEG width in pixels
  /Height vpixels                   % JPEG height in pixels
  /ImageMatrix [hpixels 0 0         % transform to image space
    vpixels neg 0 vpixels ]
  /DataSource Data                  % specify image data
  /BitsPerComponent 8               % color resolution
  /Decode [0 1 0 1 0 1]             % per red book 4.10
       >>
  image                             % call image operator
  ypos snap2 restore                % and exit
} def
```

Details on all this are in the **red book**, aka the **PostScript Reference Manual 3**.

Note that a normal PostScript transformation matrix is of form...

```
[ xscale lean climb yscale xoffset yoffset ]
```

In this case, we want to make the image hpixels wide with zero lean or climb. We want to turn the image "upside down" so that **later** image data is **further** down the page. Hence the **neg** after the vpixels scaling. Finally, we want our image bottom to be at our starting location, so we offset the entire image vertically upwards by **vpixels**.

That **/Decode [0 1 0 1 0 1]** is specific to 8-bit RGB data as per the red book.

## What Resolution?

You can use **EBAYFOTO.PDF** as a demonstration program that imports four JPEG files as thumbnails and lets you click expand them into full images.

How much .JPEG resolution is "enough"? If you do **not** downsample your .JPEG images during distillation, your images will appear the best possible, and should be click-expandable to larger sizes and still look fairly decent. But this will make your .PDF file much larger. Especially when there are several images involved.

If you **do** downsample during distillation, you can gain extremely compact files, but the images may not look (or print) quite as good initially and will fall apart on expansion. It is often better to provide detailed larger images only to those viewers who specifically want to click through on them.

If a file of mine is to be primarily viewed as a web screen, I'll usually downsample everything to 144 DPI and then provide a separate larger clickthrough image for detailed viewing. It does depend on how good printing must be and how many images are involved. The tradeoff is to get your download times minimum while preserving just enough quality to be useful.

## The "GoBack" Problem

There is one small detail remaining that may cause you grief. If you web distribute a .PDF file and click on a JPEG image on page three, you'll go to that image ok, but may return to the **start** of your document. And converting the expansion .JPG file into an **unlinked** .PDF file does not seem to help any in most browsers. This can be maddeningly infuriating if there are dozens to hundreds of pages involved.

Here are five possible workarounds to the "bad return" problem...

**Do nothing —** If the .PDF host document is only a few pages long and if there are only one or two click-expansions and if those click thrus aren't used all that much, then few of your viewers will be all that much annoyed.

**Use single page host documents —** If your host .PDF file is only one page long, then you will automatically return to the first page. Sometimes partitioning your delivery **one level up** will thus eliminate the bogus returns. This can work well where you have pages of gallery snapshots. See our **Bargain Tour** and **Its Tutorial** for further assistance. The downside is the extra host page load times.

**Open expansion in a new window —** Since the old window remains on the proper page, you automatically return to the proper place when the new window is closed or you otherwise click back. The key HTML coding is...

```
< A HREF="http:/www_usual_url" target="_blank" >
```

The downside is that many viewers do not like to have unwanted or unexpected windows opening on their screen. But ordinary .JPG images can still be used for expansion. I've yet to find out how to do this within .PDF, though.

**Use a new .PDF file with named destinations —** To do this, make your expansion image a .PDF page that is mostly the .JPEG image with an added graphic **click to return** box. One that returns to your specific page and magnification. We saw an example of this technique way on back in our older **LINKPDF1.HTML**. Additional info in the **PDFMark Reference Manual**

Ferinstance, suppose your first expandable photo is on page three of your doc. You might place and call this **destination marker** code in your PDF source...

```
/setdest1 {[/Dest /GoBack1 /Page 3 /DEST pdfmark } store
```

… and then use a URL like this for your backlink from the expanded photo **done as a .PDF file**…

```
http://webpage_name/jpg2pdf.pdf#nameddest=GoBack1
```

The crucial coding being the **#nameddest=GoBack1** that should return you. In the case of fancy pages, your **/setdest** can also alter your page view and position. That **nameddest** part appears unnecessary, for a simple and conventional **#GoBack1** seems to work as well.

Unfortunately…

> **IE sometimes PROHIBITS a .PDF file from linking to a named destination in another .PDF file. It instead may create a useless blank white screen.**

One crude hack workaround to this hassle is to have the PDF image file to be returned **from** link a .HTML **redirect** file that in turn links to the named destination. Ferinstance, your .PDF URL return coding might be…

```
http://webpage_name/iefix001.html
```

And your redirect **iefix001.html** file might look like…

```
<html>
<HEAD>
<TITLE>IE PDF RETURN BUG FIXER</TITLE>
<META HTTP-EQUIV="REFRESH" CONTENT="1;
URL=http://www.tinaja.com/glib/ebayfoto.pdf#GoBack1">
</HEAD>
<body>
Avoiding an IE Acrobat bug.
</body>
</html>
```

Note that an instant redirect is **not** a good idea as it trashes the viewer's back arrow. A minimum one second delay has been shown here.

An example utility appears as **GRABJPG.PSL** in our **PostScript** library. The disadvantages of this sledgehammer method are that extra work is involved, your "click this box to return" will be nonstandard, the redirect can be annoying, and care in destination naming will be required if one expanded image is to be able to return to several host docs.

Some ongoing **GoBack** experiments appear at **EBAYFOTO.PDF**. This method appears to work equally well with NetScape or IE.

**Use HTML image expansions —** Instead of the above expanded .PDF file followed by a redirect, you could simply use an HTML file consisting of a table holding a .JPEG image and a return URL. This limits your appearance options, and is both slower and fragmented. It does, however, eliminate the redirect delay. IE will apparently happily return to an internal .PDF page from anything **but** another .PDF file.

Other approaches to the **GoBack** problem might involve **JavaScript** or the techniques shown in **Example 8.15** of the **PDF Mark Reference Manual**. Also asking your viewers to open .PDF in a new window rather than inside a browser can also possibly be of help. Please **email me** if you have any better solutions to the **GoBack** problem.

## For More Help

Additional background along with related utilities and tutorials appears on our **GuruGram**, **PostScript**, and **Acrobat** library pages.

Consulting assistance on any and all of these and related topics can be found at **http://www.tinaja.com/info01.asp**. As can our image development and processing services.

Additional **GuruGrams** columns await your ongoing support as a **Synergetics Partner**.