Table of Contents

The Acrobat User

PDF/X and PDF/A
Unnecessary mystery surrounds these CGATS standards. They are really just normal PDF files that agree to certain restrictions.

PostScript Tech

Global VM
Global VM is a part of virtual memory that is not subject to the *save* and *restore* operators. Why would you ever want to bypass these important memory management operators? This month we'll see.

Class Schedule

April, May, June, July

What's New?

PDF Classes Overtake PS Classes
More students signed up for PDF classes than for PS classes in the past year.

Contacting Acumen

Telephone number, email address, postal address

Journal feedback: suggestions for articles, questions, etc.

PDF/X and PDF/A

A certain mystery seems to surround PDF/X and PDF/A in many people's minds. Among other things, I am often asked that you need to add to a normal PDF file to make it PDF/X or PDF/A and whether the resulting file can still be opened by Acrobat or Mac OS' *Preview* application.

This month, let us look at these the purposes and details of these two standards. It turns out that PDF/X and PDF/A files are subsets of the standard PDF specification; a PDF/X or PDF/A file is just a standard PDF file that agrees to abide by certain restrictions.

The two standards are intended for different purporses. PDF/X is intended for pre-press; PDF/A is intended as a long-term archive format. Let's look at them in detail.

Next Page ->>

PDF/X

The goal of the PDF/X specification is to eliminate all of the problems that occur with PDF files intended for pre-press. If you will be sending a document to a print shop as a PDF file, sending it as a PDF/X file will ensure that all of the errors that routinely keep a PDF file from printing will have been eliminated.

The PDF/X standard defines several variants, of which only two are used by anyone: PDF/X-1a and PDF/X-3. The latter is an extension of the former, so let's start with PDF/X-1a.

PDF/X-1a

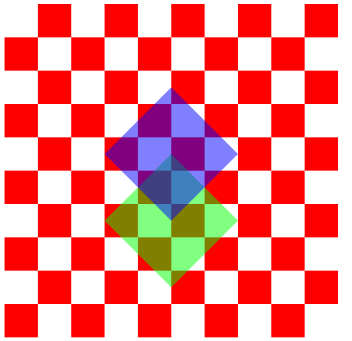
A PDF/X-1a file is just a PDF file that abides by the following restrictions:

Acrobat 4 compatible

PDF/X-1a files must be compatible with PDF 1.3.

A PDF/X-1a file must be compatible with Acrobat 4, which implements PDF version 1.3. At issue here is transparency; this restriction eliminates one of the few major (and irreconcilable) differences between PDF and PostScript.

The PDF file format includes support for transparency (well, opacity, actually); objects on a PDF page may allow the background to show through. Files in prepress are typically printed on PostScript devices and PostScript, unfortunately, has absolutely no support for transparency; painting in PostScript is always opaque, completely blocking out the background.



Next Page ->

As a consequence of this incompatibility, transparent objects in a PDF file will be printed opaque on a PostScirpt device.

To avoid this problem, a PDF file conformant with PDF/X-1a will not use transparency.

Self-contained

PDF/X-1a files must be self-contained.

A PDF/X-Aa file may have no references to items outside of the PDF file itself.

Primarily, this means that all fonts must be embedded in the PDF file.

Colorspace restrictions

PDF/X-1a files may express colors only using CMYK, gray, or spot colors.

This eliminates most of the common problems associated with printing a PDF file on a press. The file may also use color tables (such as in an 8-bit image) that are based on the above colorspaces.

Technically, the PDF/X-1a file may use any of the following PDF colorspaces: DeviceCMYK, DeviceGray, Separation, Indexed. The Indexed colorspace (which specifies color in terms of a color table) must be based on one of the other three colorspaces.

In particular, note that a PDF/X-1a file may not have managed colors, based on ICC or other profiles.

The file does need to provide an *output intent,* which indicates the paper or other medium for which the document is intended. This output intent gives precise meaning to the color specifications in the PDF file.

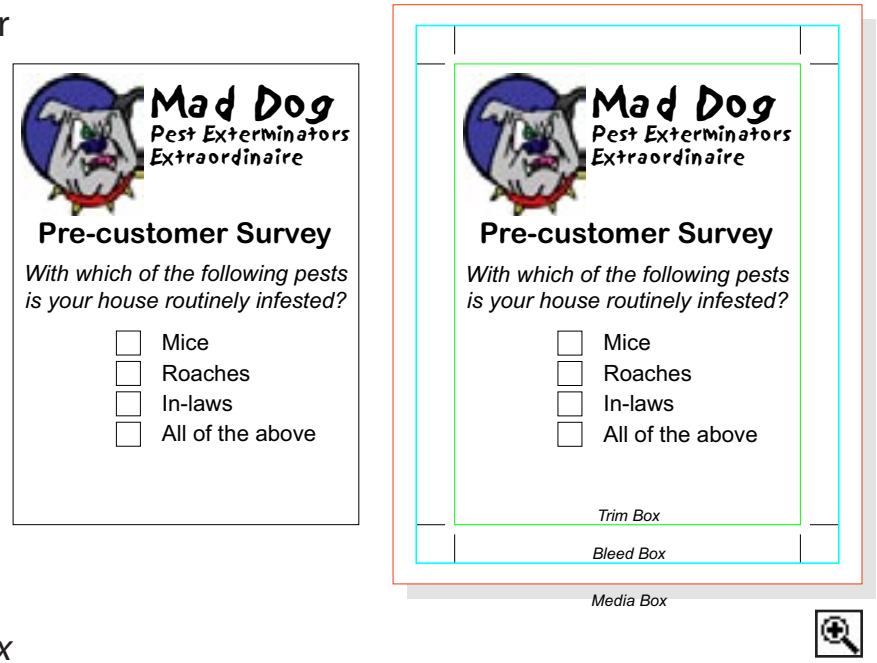
Next Page ->

Three boxes required

PDF/X-1a files must contain MediaBox and an *ArtBox entries for each page.*

The PDF file specification defines several rectangular regions ("boxes") that may be associated with each PDF page. The *MediaBox* is required in a PDF file; the *ArtBox* is informational only, though a PDFviewer may choose to use them for its own purposes. The two boxes required by PDF/X-1a are:

- *MediaBox* - This is the rectangular area occupied by the physical medium on which the page will be printed. In effect, this defines the papaer size for the page.
- *Art Box* - This rectangle defines the area occupied by all the marks on the document page. This area excludes crop marks and other marks that are not part of the final, trimmed document page.



The PDF spec also defines a *BleedBox* (which enloses all marks on the page, including crop marks) and *CropBox* (which defines the part of the PDF page that should actually be printed). These two rectangles are not required by the PDF/X spec, although they are allowed to be present in the file.

Next Page ->

Acumen Journal: Acrobat User 5

No encryption/passwords

PDF/X-1a files must not be encrypted or password protected.

The PDF file must be accessible without any security hindrance. If security is important to you, then you must create a PDF file that is not conformant with PDF/X-1a.

No halftone screens or transfer functions

PDF/X-1a files must not specify a halftone screen or a transfer function

The presumption is that an appropriate screen will be set up at the printer or press.

A transfer function specifies a "gamma function," that adjusts the grays (and colors) specified in the PDF file to values appropriate for the press on which the file is being printed. As with halftone screen, thsi is something better specified by the press folks at the print shop.

Compression limitations

Data within a PDF/X-1a file may have only the following compression applied to it: JPEG, flate, run length, CCITTFax.

The PDF file specification defines a relatively large collection of compression methods that may be applied to image and other data within a PDF file. The PDF/X-1a spec restricts the allowed compression methods to those listed above.

This requirement looks to me as though it's a part of being compatible with PDF-1.3.

Next Page ->

Acumen Journal: Acrobat User 6

Alternative images *If an image has an alternative, it must be identical to the original image.*

The PDF file spec allows every image in a document to have one or more alternative versions (perhaps a high-res version to be used in printing, for example). In a PDF/X-1a document, the alternative versions of an image must be identical to the original. (This makes it seem to me pointless to have an alternative image at all.)

Trapping Identified *The presence or absence of trapping must be stated.*

A PDF file may have a *Trapping* value in it that indicates whether that file is trapped. This value is required in a PDF/X file. It must be either *true* (if the file is trapped) or *false* (if the file is not trapped).

Comment Note that how much uncertainty has been eliminated by the PDF/X-1a standard. A PDF/X-1a file can be opened with any version of Acrobat dating from the past five years; it is guaranteed to have relatively simple color specifications; it will not depend upon any fonts or color profiles being installed on the system; it will be openable without passwords or other decryption. The printer operator will have control over the halftoning and gamma of the printer without the PDF file attempting to override the printer's settings.

So what does PDF/X-3 add to the PDF/X-1a spec?

Next Page ->>

PDF/X-3

PDF/X-3 is identical to PDF/X-1a, with a single important exception: the file may have managed colors in it, in addition to the CMYK, gray, and spot colors allowed by PDF/X-1a. There do exist some well-calibrated environments that use managed colors (using, for example, ICC-based color profiles); PDF/X-3 gives them the ability to do so while still adhering to a pre-press standard.

PDF/A

PDF/A has a different purpose than PDF/X. A PDF/A file is intended for archival purposes; this standard intends to make the file readable for as long a time as possible.

Here are the rules:

Self-contained

A PDF/A file must be self-contained.

As with a PDF/X file, a PDF/A file must contain no references to resources that reside outside of the file. This means that fonts, profiles, etc. must be embedded in the PDF file.

Managed Color Only

A PDF/A file must contain only CIE-based managed colors.

Unlike in a PDF/X file, *all* of the colors within a PDF/A file must be managed. If the file contains RGB, CMYK, or grayscale values, they must be accompanied by an ICC profile (bundled as part of something called an *output profile)* that identifies the characteristics of the particular inks, phosphors, liquid crystal diodes you have in mind.

Next Page ->

Acumen Journal: Acrobat User 8

Image restrictions

A PDF/A file may not have alternative or interpolated images.

Alternative images, as described earlier, are not allowed in a PDF/A file. Also forbidden is image interpolation, a technique whereby the PDF viewer attempts to smooth the border between adjacent pixels within the image.

No halftone screens or transfer functions

PDF/A files may not specify either halftone screens or transfer functions.

As with PDF/X, a PDF/A file may not specify either a halftone screen or a transfer function.

No OpenType Fonts

PDF/A files may not contain OpenType fonts.

Fonts may be Type 1, TrueType, or Type 3. In all cases, they must be embedded in the PDF file of course.

No transparency

A PDF/A file may not specify the opacity of painted objects.

As we said earlier, transparency can offer significant problems to a printer.

Viewer Compatibility

As you can see from this description, both PDF/X and PDF/A files are completely compatible with all PDF viewers. They differ from standard PDF files only in avoiding certain problematic, though completely legal, PDF characteristics.

Next Page ->

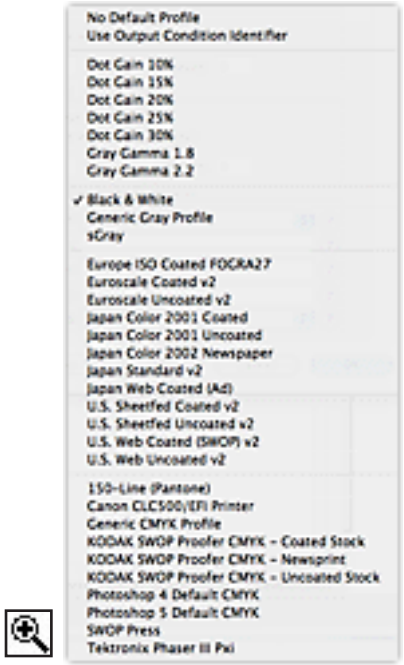
Acumen Journal: Acrobat User 9

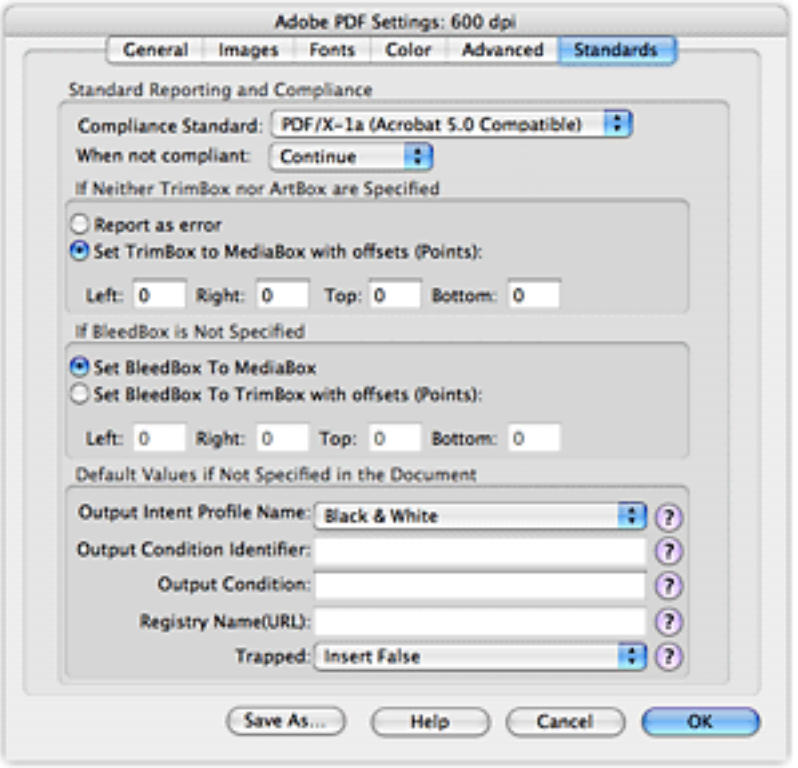
Making Compliant PDF Files

Acrobat Distiller can make files compliant with either PDF/X or PDF/A; to do so, the PostScript code that it receives must obey the above rules. Among the Distiller PDF Settings is a *Standards* tab that lets you specify that Distiller should make a PDF file that is PDF/X or /A conformant.

Most of these controls are reasonably self-evident: which standard the PDF file should follow; what should happen if the required "boxes" are missing, etc.

Note that you need to pick the output intent from a list; you should pick the item that most closely matches the paper on which your job will be printed.





Return to Main Menu

Global VM

Among PostScript programmers, managing virtual memory is somewhat obsessive behavior. As you recall from your PostScript classes (as well as all your PostScript experience since then), virtual memory is the RAM that is directly accessible to your PostScript program (as opposed to the memory reserved for page buffer, stacks, etc.).

We directly manage VM with the PostScript operators *save* and *restore*. The *restore* operator returns memory use (among other things) to whatever it was when you did the corresponding *save*. Usually, we enclose successive blocks of our PostScript code in *save/restore* pairs, most often putting a *save* at the beginning and a *restore* at the end of each page.

PostScript Level 2 introduced *Global VM,* a separate part of VM that is not subject to *save* and *restore*. The memory used by composite objects (string, arrays, dictionaries) created in Global VM will not be freed when you execute a *restore*.

This doesn't sound very useful, except perhaps for the purpose of implementing memory leaks. In fact, Global VM is rarely something you want to use or pay attention to; it was originally invented for Display PostScript to address a circumstance that doesn't occur in printer PostScript.

Nonetheless, there is one case in which you do want to allocate your strings, etc. in Globa VM: resource definitions that are stored on a printing device's hard disk.

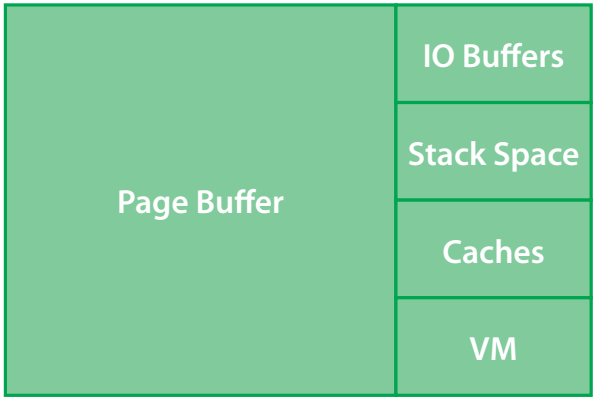
Let's look at this in detail.

Next Page ->

Acumen Journal: PostScript Tech 11

VM: A Review

The RAM associated with a PostScript device is put to a number of purposes: page buffers, I/O buffers, stack space, a variety of caches. The RAM that is actually available for executing your PostScript code is whatever remains after these other items are allocated. This is virtual memory, VM. This is where your string, arrays, and dictionaries (including fonts) are stored; ninety-nine times out of ten, when you run out of memory, it is VM that you have exhausted.



Garbage Collection

Unfortunately, as the diagram above illustrates, VM is a small part of the total RAM in the printer, so it is relatively easy to fill it up.

PostScript Level 2 introduced automatic garbage collection to virtual memory, so nowadays, when VM approaches full, PostScript's memory management can scan through VM, freeing the memory used by inaccessible strings, arrays, and dictionaries.

Automatic garbage collection can be serious time consumer; there is nothing more annoying than having your high-speed printer come to an abrupt halt and remain seemingly idle for a minute or two while the automatic garbage collector peruses memory, looking for reclaimable items.

Hence, *save* and *restore*.

Next Page ->

Acumen Journal: PostScript Tech 12

save & restore

These two operators save and restore the state of, among other things, virtual memory. The *save* operator does some bookkeeping and leaves the resulting data on the stack as a *saveobject*.

What else they do

The *save* and *restore* operators actually save and restore the following:

- VM
- Key-value pairs
- The Graphics State stack

In fact, to a good extent, *save* and *restore* affect everything except the operand, dictionary, and execution stacks and the caches.

The *restore* operator takes a saveobject from the stack and resurrects the earlier state of VM. Among other things, this returns the amount of VM used to its earlier level.

Save and restore taken together are much quicker than the automatic garbage collection reoutines. By enclosing appropriate parts of your PostScript code in save/restore pairs, you can ensure that the automatic garbage collection never is invoked by your PostScript code.

Thus, in the following code:

```
save
/s (Howdy) def
restore
```

The restore operator reclaims the VM used by the string *(Howdy),* as well as destroying the key-value pair *s*. (Since neither the string nor the key-value pair existed when you did the *save,* they both are reclaimed by the *restore.*

Global VM

Global VM, as I said earlier, is a new class of virtual memory that is not subject to *save* and *restore*. Had our *Howdy* string been allocated in Global VM, the *restore* operator would not have reclaimed the memory it occupied.

Next Page ->

Acumen Journal: PostScript Tech 13

Ignoring for the moment why you would want to allocate a string in Global VM, *how* to you allocate a string in Global VM?

setglobal

By default, PostScript defines string, arrays, dictionaries, etc. into normal VM, which is referred to in this context as *Local VM*. You change the location in which new objects are allocated with the *setglobal* operator.

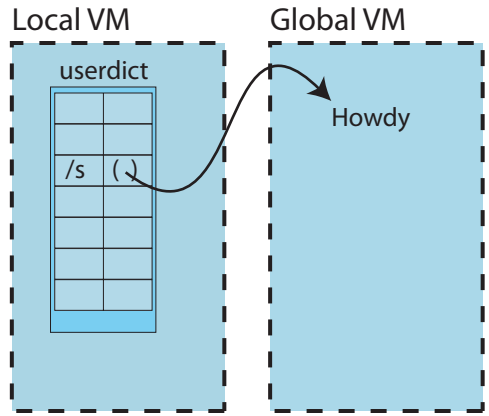
```
true setglobal
```

This operator takes a boolean value from the operand stack and sets the current allocation mode to global (true) or local (false).

Let us modify our previous example to set the allocation mode to *global* before defining our key-value pair:

```
true setglobal
save
/s (Howdy) def
restore
```

The string *Howdy* would now be allocated in global VM, as illustrated at right.



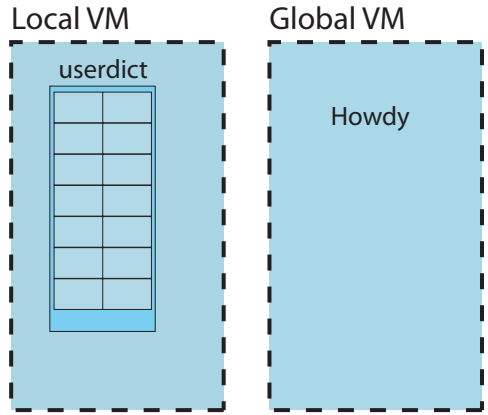
true setglobal save /s (Howdy) def

Our *Howdy* string would not have been reclaimed by the *restore* operator; it would have still resided in memory, taking up VM until eventually scavenged by the automatic garbage collection routine.

Next Page ->

Interestingly, the key-value pair of which *Howdy* was a part (whose key was *s) has* been reclaimed. The *def* operator put this key-value pair into *userdict,* which is allocated in local VM and is, therefore, subject to *save/restore.*

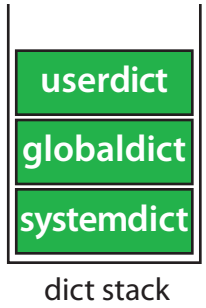
The memory situation after the *restore* is illustrated at right; note that *Howdy* still occupies VM, even though the key-value pair that contained it is gone from userdict. The string *Howdy* is now inaccessible to our PostScript program and is therefore subject to automatic garbage collection.



restore

globaldict

If we want key-value pairs to survive a *restore,* we must place them in a dictionary that is itself allocated in global VM. PostScript supplies such a dictionary on the dictionary stack: *globaldict,* sandwiched in between systemdict and userdict. Globaldict, like userdict, is a writeable dictionary intended for your program's key-value pairs; however, since globaldict is allocated in global VM, key-value pairs placed into it will survive across *restore* executions.



dict stack

We can modify our earlier code so that it places globaldict on top of the dictionary stack:

```
true setglobal
globaldict begin
save
/s (Howdy) def
restore
```

Next Page ->

Acumen Journal: PostScript Tech 15

In this new version of the code, our variable *s* is defined into globaldict, which became our current dictionary with the *begin*. As a consequence, the variable *s* will still be available after the *restore* invocation.

Allocation Errors

The only misstep possible when working with global VM is trying to put a locally-allocated object into a globally-allocated object. For example, consider the following code:

```
globaldict begin
save
/s (Whoop-de-doo) def
restore
```

We place globaldict on the dictionary stack, do a save, and then define a string variable into globaldict.

Consider what must happen when we execute *restore*. Because the key-value pair */s* is defined into globaldict, it must survive across the *restore*. However, our VM allocation mode throughout this snippet is *local;* thus, the string is allocated in local VM and must therefore be reclaimed by the *restore*.

The restore, scavenging the string, would leave an invalid pointer in the globally-allocated key-value pair */s*. PostScript will not allow this and will return a PostScript error when you try to execute *def*. You are not allowed to store a locally-allocated object (the string, in our case) into a globally-allocated object (globaldict).

The exact error you get seems to vary with interpreter. I used to see *VM allocation* errors; I notice the current version of Distiller yields an *invalidaccess* error.

Next Page ->

But, Why?

Alright, so we have established how to tell PostScript to allocate strings, arrays, and dictionaries in global VM. The question remains: why would we want to do that?

The answer is: you wouldn't; at least, not often. The only circumstance I have encountered is a result of the way that fonts and other resources are stored on a printer's hard disk.

Disk-based Resources

It is useful in many environments to store commonly-used fonts on a printer's hard disk so that they don't need to be downloaded with every print job. When you do this, what you are actually storing on the printer's disk is a PostScript program that, when executed, creates the font resource. This "font program" creates a disctionary, puts into the dictionary everything needed for a font, and then converts that dictionary into a font with the *definefont* operator:

```
%!PS-AdobeFont-1.0
10 dict begin
/FontType 1 def
/FontMatrix [ .001 0 0 .001 0 0 ] def
...
...
/Optima currenctdict end definefont
```

The call to *definefont* is usually encrypted and looks like ASCII-encoded nonsense.

Next Page ->

Acumen Journal: PostScript Tech 17

Using Disk-based Resources

When you execute the *findfont* operator, that operator looks for the font in memory (specifically, in the *FontDirectory* dictionary). If it doesn't find the font in memory, it executes the font's file on the hard disk, which places the font in the Font Directory; the program can now use the font.

Unfortunately, in commercial-grade output, *findfont* is usually executed in the PostScript code associated with a page in the document, which is usually enclosed by *save* and *restore.* As a result, at the end of the page, the *restore* will cause the font dictionary to be reclaimed. If the same font is used on the next page, *findfont* will need to hit the hard disk again to re-execute the font program.

Resources in Global VM

To ensure that once a resource is stored in VM it stays there, font programs and other resource definitions always place their resulting definitions into global VM:

```
%!PS-AdobeFont-1.0
true setglobal
10 dict begin
/FontType 1 def
/FontMatrix [ .001 0 0 .001 0 0 ] def
...
...
/Optima currenctdict end definefont
false setglobal
```

Next Page ->

Acumen Journal: PostScript Tech 18

Since the resource dictionary and all of its contents are being allocated in global VM, they will not be reclaimed at the end of the page and will still be available to the next page's *findfont* or *findresource*.

Thus, we need hit the disk for a resource only the first time we fetch that resource.

This results in a minor, but real reduction in execution time.

Have You Used Global VM?

I've never found any other case where using global VM was the best solution to a problem. If you have ever used global VM to solve a problem, I'd be interested to hear about it. Let me know the circumstances by email, if you would.

Return to Main Menu

Schedule of Classes, April-June 2006

Following are the dates of Acumen Training's upcoming PostScript and PDF Technical classes. Clicking on a class name below will take you to the description of that class on the Acumen training website.

These classes are taught in Orange County, California and on corporate sites world-wide. See the Acumen Training web site for more information.

Technical Classes

	PDF File Content and Structure 1	May 15–18		July 10-13
NEW!	PDF File Content and Structure 2	May 1–4		
	PostScript Foundations	May 8–12		
	Variable Data PostScript		June 5–9	
	Advanced PostScript			
	PostScript for Support Engineers		June 19-23	

Course Fee The PostScript and PDF classes cost $2,000 per student.

Registration Info

Acrobat Class Schedule

Regretfully, I have suspended teaching Acrobat classes.

Return to Main Menu

Contacting John Deubert at Acumen Training

For more information

For class descriptions, on-site arrangements or any other information about Acumen's classes:

Web site: http://www.acumentraining.com **email:** john@acumentraining.com

telephone: 949-248-1241

mail: 24996 Danamaple, Dana Point, CA 92629

Registering for Classes

To register for an Acumen Training class, contact John any of the following ways:

Register On-line: http://www.acumentraining.com/registration.html

email: registration@acumentraining.com

telephone: 949-248-1241

mail: 24996 Danamaple, Dana Point, CA 92629

Back issues

All issues of the *Acumen Journal* are available at the Acumen Training website:
http://www.acumenjournal.com/AcumenJournal.html

Return to First Page

What's New at Acumen Training?

PDF Classes Overtake PS

Nothing much to report this time. I do note that over the past year, I have taught more PDF than PostScript classes. Presumably this trend will continue, with more and more printer and printing organizations moving to PDF for their print streams. This is sad, because after twenty years in the business, I still think PostScript is a lot of fun to play with.

Return to First Page

Journal Feedback

If you have any comments regarding the *Acumen Journal,* please let me know. In particular, I am looking for three types of information:

Comments on usefulness. Does the Journal provide you with worthwhile information? Was it well written and understandable? Do you like it, hate it? Did it induce dreams involving your childhood pet, Fluffy, who was inexplicably able to talk.?

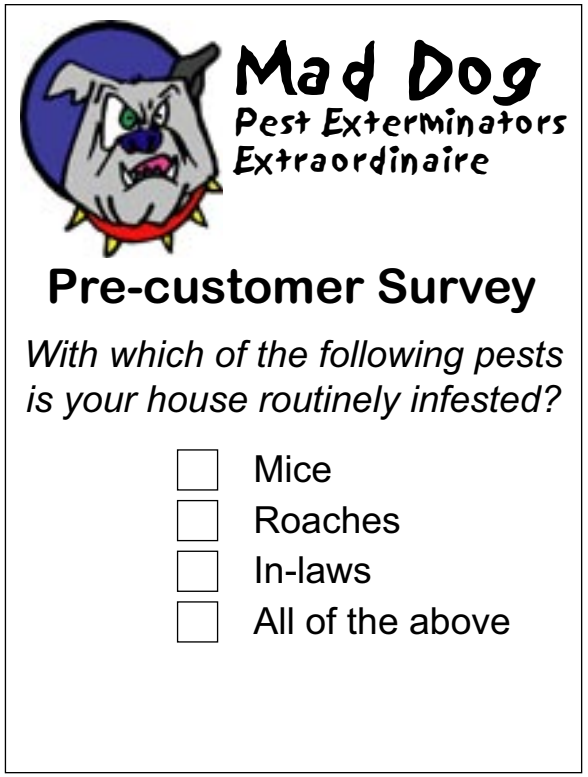
Suggestions for articles. Each Journal issue contains one article each on PostScript and Acrobat. What topics would you like me to write about?

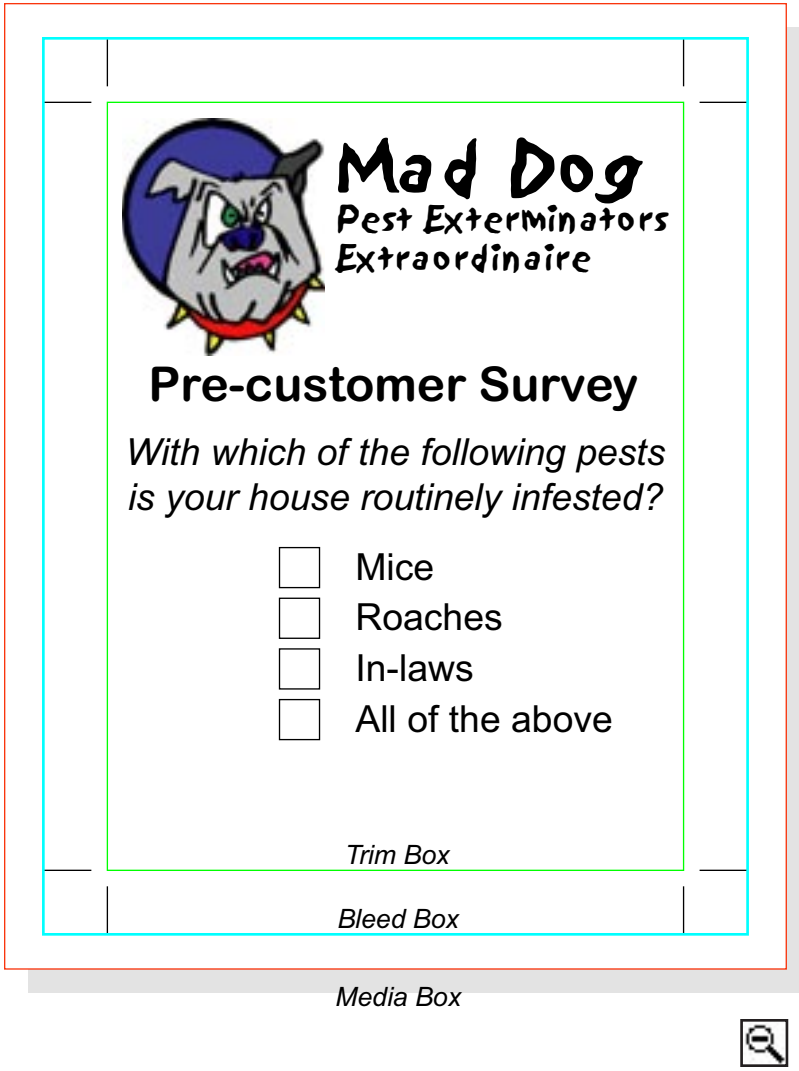
Questions and Answers. Do you have any questions about Acrobat, PDF, or PostScript? Feel free to email me about. I'll answer your question if I can. (If enough people ask the same question, I can turn it into a Journal article.)

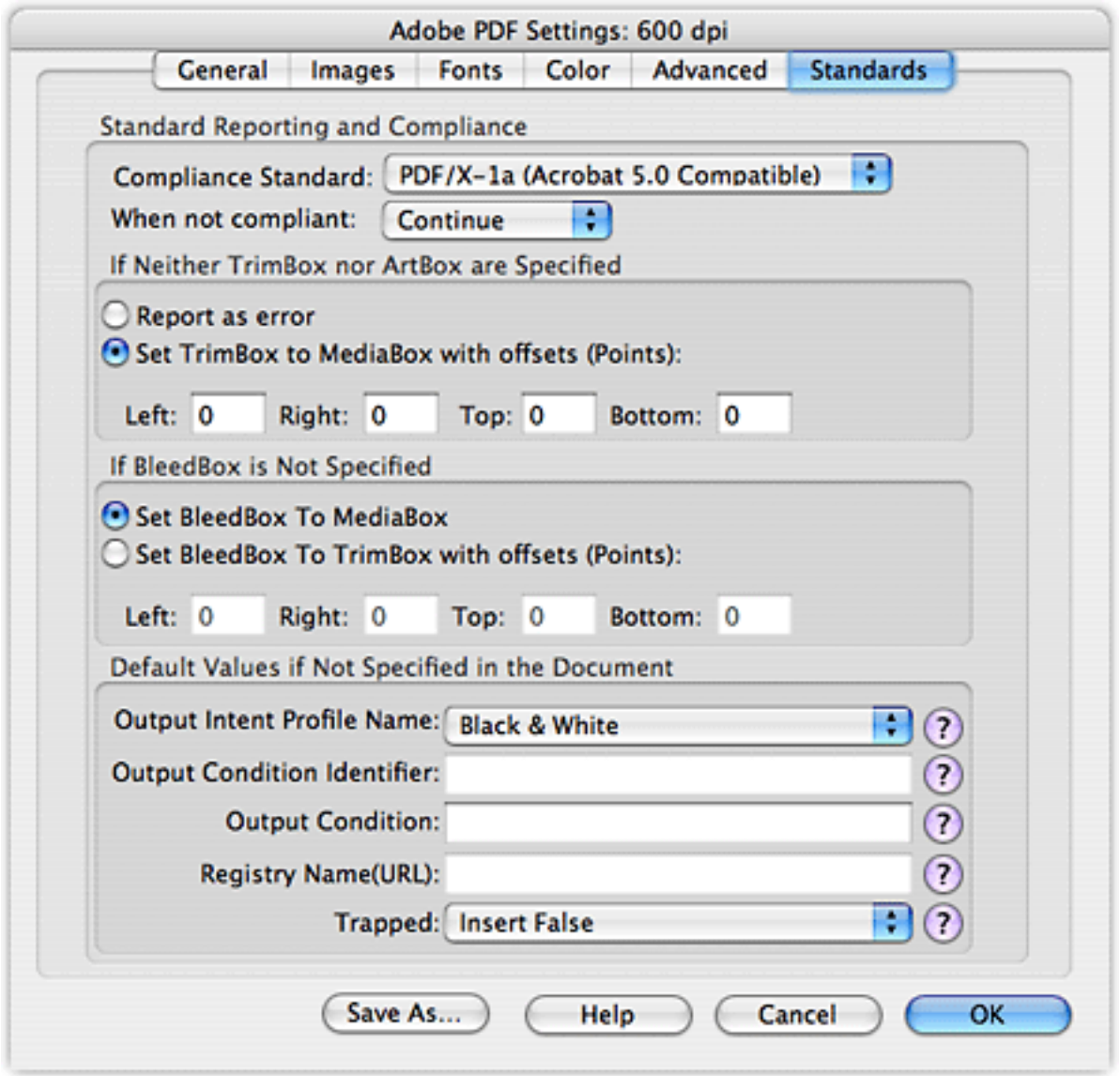
Please send any comments, questions, or problems to:

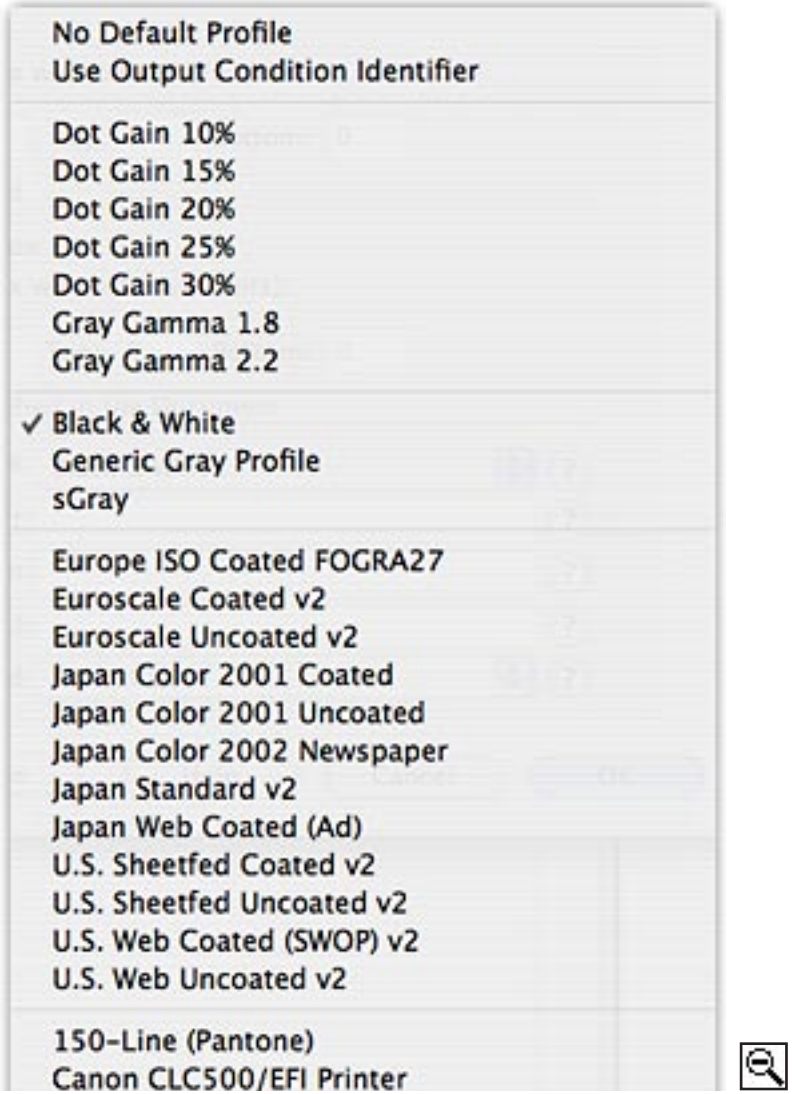
journal@acumentraining.com

Return to Menu


Mad Dog
Pest Exterminators
Extraordinaire
Pre-customer Survey
With which of the following pests is your house routinely infested?
Mice
Roaches
In-laws
All of the above


Mad Dog
Pest Exterminators
Extraordinaire
Pre-customer Survey
With which of the following pests is your house routinely infested?
Mice
Roaches
In-laws
All of the above
Trim Box
Bleed Box
Media Box


Adobe PDF Settings: 600 dpi
General
Images
Fonts
Color
Advanced
Standards
Standard Reporting and Compliance
Compliance Standard:
PDF/X-1a (Acrobat 5.0 Compatible)
When not compliant:
Continue
If Neither TrimBox nor ArtBox are Specified
Report as error
Set TrimBox to MediaBox with offsets (Points):
Left: 0
Right: 0
Top: 0
Bottom: 0
If BleedBox is Not Specified
Set BleedBox To MediaBox
Set BleedBox To TrimBox with offsets (Points):
Left: 0
Right: 0
Top: 0
Bottom: 0
Default Values if Not Specified in the Document
Output Intent Profile Name:
Black & White
Output Condition Identifier:
Output Condition:
Registry Name(URL):
Trapped:
Insert False
Save As...
Help
Cancel
OK


No Default Profile
Use Output Condition Identifier
Dot Gain 10%
Dot Gain 15%
Dot Gain 20%
Dot Gain 25%
Dot Gain 30%
Gray Gamma 1.8
Gray Gamma 2.2
✓ Black & White
Generic Gray Profile
sGray
Europe ISO Coated FOGRA27
Euroscale Coated v2
Euroscale Uncoated v2
Japan Color 2001 Coated
Japan Color 2001 Uncoated
Japan Color 2002 Newspaper
Japan Standard v2
Japan Web Coated (Ad)
U.S. Sheetfed Coated v2
U.S. Sheetfed Uncoated v2
U.S. Web Coated (SWOP) v2
U.S. Web Uncoated v2
150-Line (Pantone)
Canon CLC500/EFI Printer