Table of Contents

Acumen Training

Acumen Journal, © 2001 John Deubert, Acumen Training

Making PostScript for PDF

Behind every successful PDF file is a successful PostScript file. If you are making professional-grade PDF files, you are almost certainly creating a PostScript file and handing it to *Acrobat Distiller, PDF Creator,* or their equivalent.

What about that PostScript file? What settings should you choose in your Print dialog box so that your PostScript file will convert into a successful PDF file? Fonts embedded or not? How should TrueType fonts be handled? Binary or ASCII?

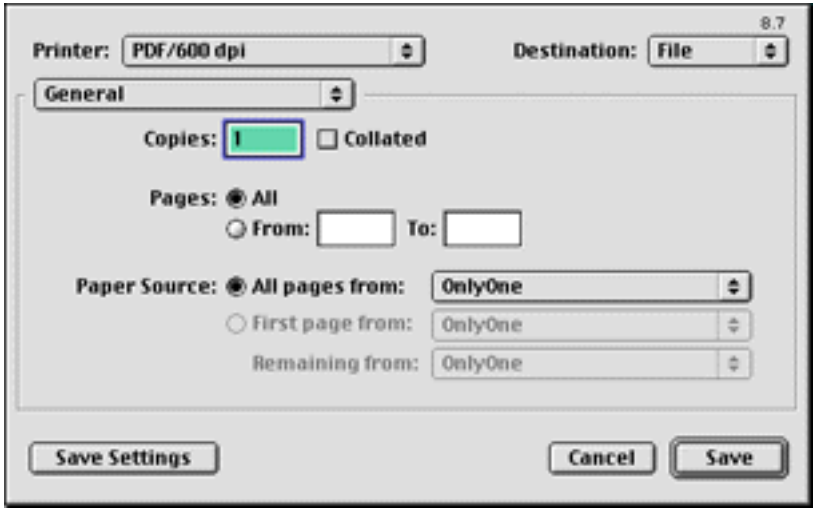
This month and next, The Acrobat User looks at how to make a PostScript file appropriate for handing to Distiller. We'll look at the Mac and Windows print controls and also look at QuarkXpress' print settings.

This month, we'll look at the Macintosh and QuarkXpress. We'll pick up Windows' controls, which are a bit more extensive, next month.

Next Page →

Making PostScript on the Mac

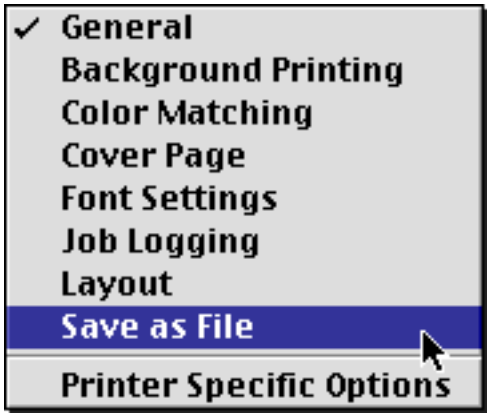
When you select *Print* from your Macintosh application, you are faced with the familiar dialog box at right.



To print your document to a PostScript file, you simply select *File* from the *Destination* menu.

When you're making a PostScript file for PDF, you need to pay attention to some of the controls accessible through the Print dialog box's main pop-up menu.

In particular, you want to go to the *Save as File* controls.



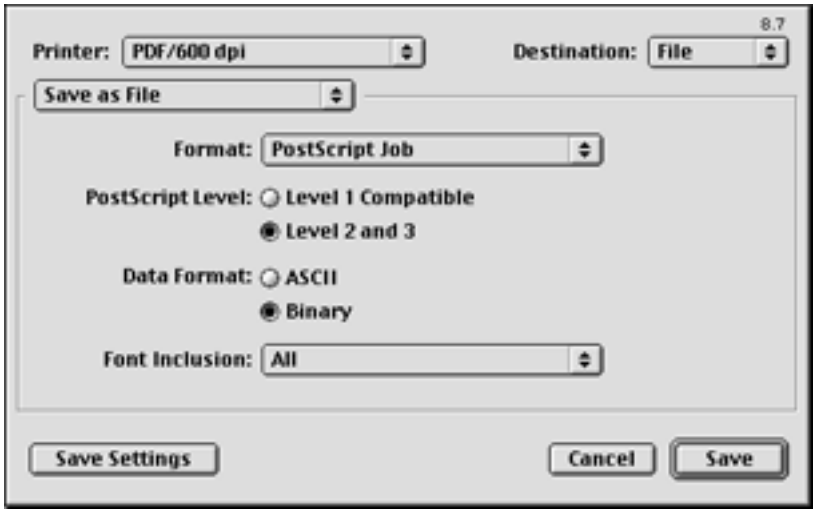
Next Page →

Save as File

This set of controls dictate the characteristics of the PostScript file created by the Macintosh print driver. The Mac pays attention to these controls only if we have selected *file* for the destination of the print job.

There are four sets of controls here; lets see how we should set each of them.

Next Page →



Format This menu specifies to what kind of file this document should be printed. You can choose to save the document as a PostScript job, three flavors of EPS and directly as a PDF file.

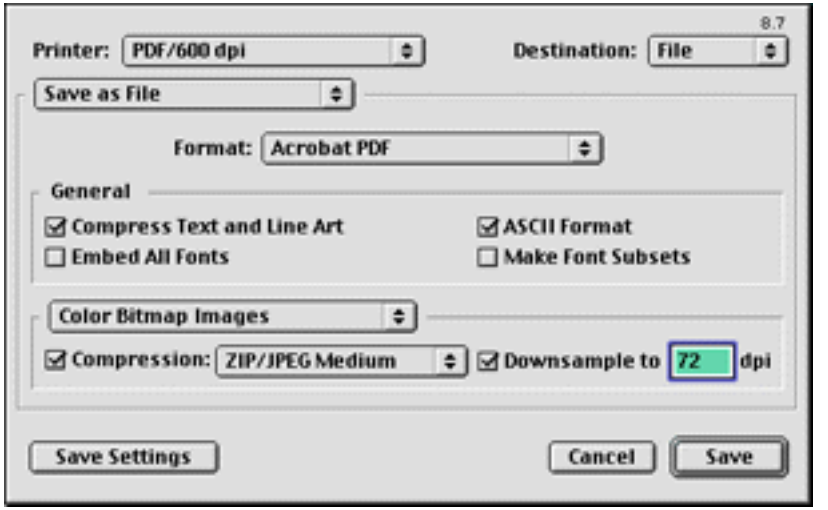
You should pick *PostScript Job.* This tells the print driver to save to disk a PostScript file appropriate for sending to a printer.

The EPS choices should be avoided because Encapsulated PostScript files are not intended for sending directly to a print device; they're designed to be embedded in some other document. As a consequence, they may not have fonts embedded, they may not specify a page size, they may not even generate a page. Some or all of these weaknesses will roll over into the PDF file.

Next Page →

What about "Acrobat PDF?"

The *Acrobat PDF* choice is tempting. If you choose this format, the dialog box's controls change to a subset of the Distiller job options.



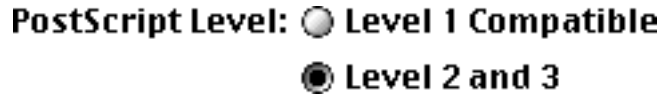
When you click the *Save* button, the print driver will create a PostScript file, launch Acrobat Distiller, set the job options as you have specified, and tell Distiller to convert the PostScript file to PDF.

Unfortunately, the subset of controls is too small. There are several very important job option controls that don't appear here. In most circumstances, you will be safer making a PostScript file and explicitly converting that to PDF.

Next Page →

PostScript Level

This pair of radio buttons allow you to choose what set of PostScript language features may be used in the PostScript file.



If you are converting this file to PDF, you should pick *Level 2 and 3.* All other things being equal, a PostScript Level 2 or 3 file will be smaller and execute faster than the Level 1 equivalent. Since all current PostScript-to-PDF converters understand at least PostScript Level 2, there is no good reason for picking *Level 1 Compatible.*

Data Format

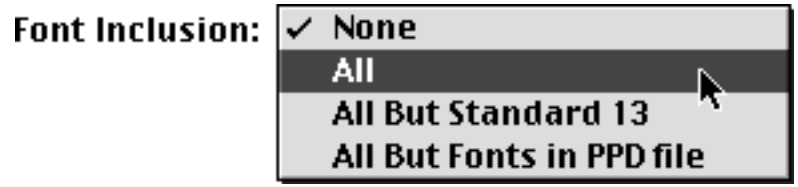
These radio buttons affect the format in which image data and, in some circumstances, fonts are embedded in the PostScript file. If you have worked with PostScript files for a long time, you tend to think of ASCII as large but safe and binary as more compact but a little risky.

This doesn't apply to PostScript files for PDF. Choose *Binary.* No PDF creator has any trouble with binary data in PostScript files.

Next Page →

Font Inclusion

This final pop-up menu asks what fonts you want to have embedded in the PostScript file.



The correct answer is: *All.*

The only way to ensure that your fonts are correctly embedded in the final PDF file is to embed them in the PostScript.

Actually, Distiller will embed Type 1 fonts into the PDF file if they are installed on your computer system, regardless of whether they are embedded in the PostScript file. Jaws Systems' *PDF Creator,* on the other hand, wants Type 1 fonts embedded in the PostScript file.

TrueType fonts must be embedded in the PostScript if they are to be embedded in the PDF file.

All this to say: choose *All.*

That's It

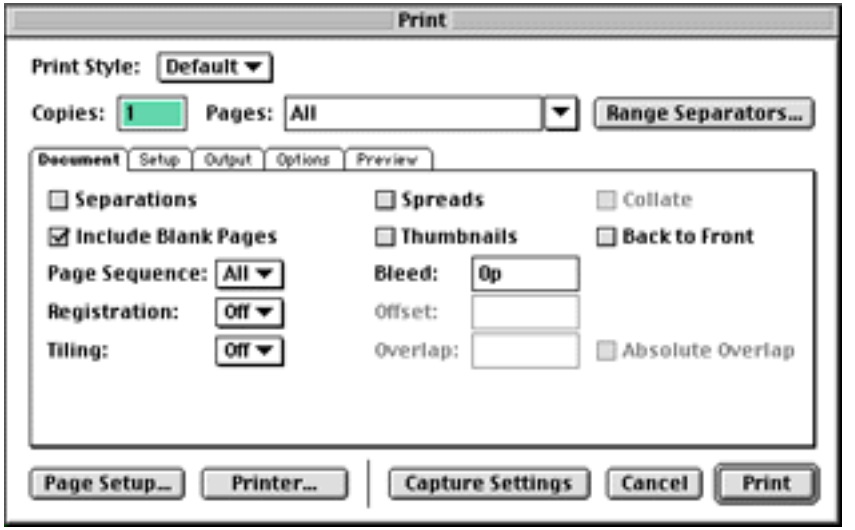
Those are all the controls in the Macintosh print driver that affect the final PDF files you create. However, many of us use applications such as QuarkXpress and PageMaker to create our PDF documents. These applications give us controls in addition to the standard print drivers'.

We'll take a look at the controls for QuarkXpress. These are typical of the controls supplied by such high-end applications.

Next Page →

QuarkXpress® Controls

When you select *Print* from the QuarkXpress File menu, you are faced with the multi-tab dialog box at right.



To set up QuarkXpress for making your PostScript file, you will need to visit three places in this dialog box:

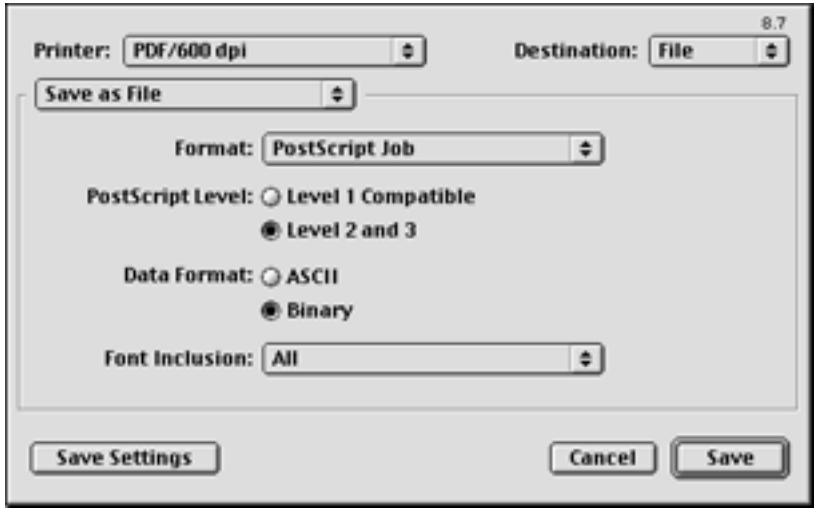
- Click on the *Printer…* button and set the standard Macintosh controls.
- Go to the *Setup* tab and specify a PPD file.
- Visit the *Options* tab and specify ASCII vs. Binary, OPI image omission, etc.

Let's look at these in detail…

Next Page →

***Printer…* Button**

Clicking on the *Printer…* button, you get access to the standard Macintosh Print dialog box. You need to go to the *Save as File* controls and set these as we discussed earlier:



- Format: PostScript Job
- PostScript Level: Level 2 & 3
- Data Format: Binary
- Font Inclusion: All

And, oh, by the way:

- Destination: File

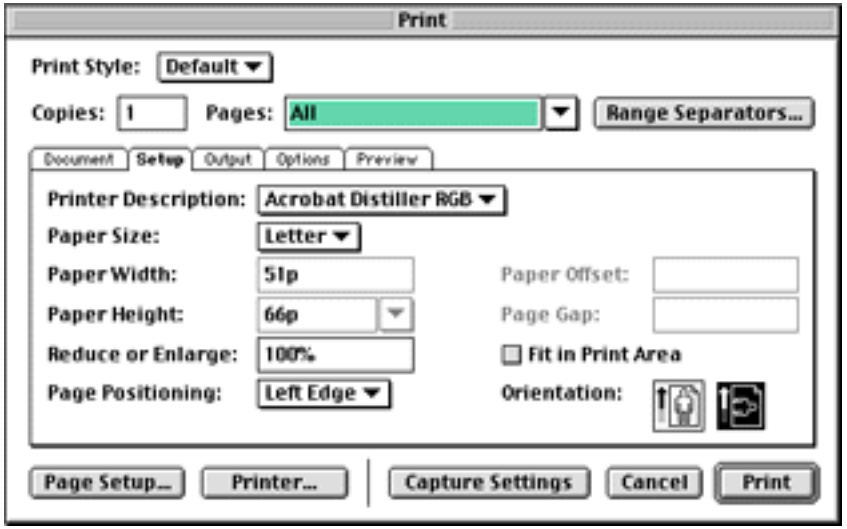
This last is necessary if QuarkXpress is to send its output to a file.

When you click on the *Save* button, you will be asked for a name for the PostScript file and then returned to QuarkXpress' Print dialog box.

Next Page →

***Setup* Tab**

In the *Setup* tab, there are only two controls to which you need to pay much attention when making a PDF-bound PostScript file:



Printer Description

Here you select the PPD file that QuarkXpress will use to generate its PostScript. The best choice here is any Acrobat Distiller PPD file.

(Acrobat installs this PPD file in your Printer Descriptions folder upon installation.)

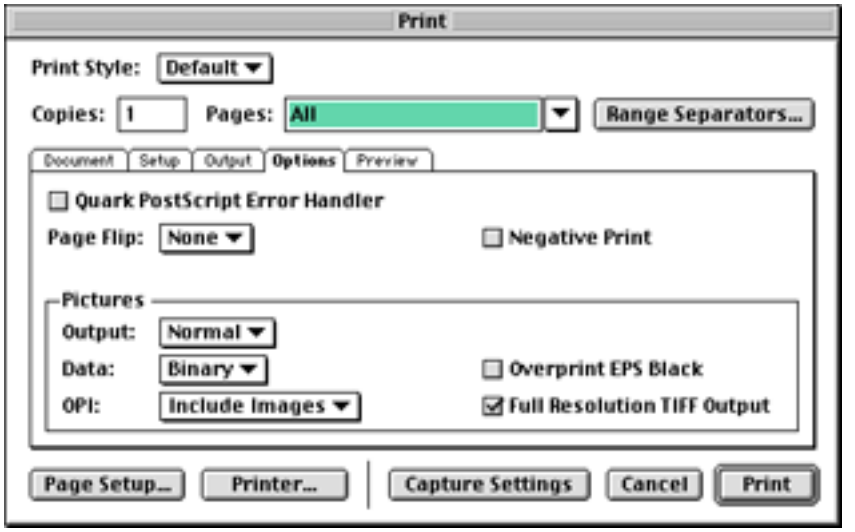
This is especially important if your document has color in it; QuarkXpress replaces all color with grays if it thinks it is printing to a black-and-white device. Distiller and its cousins all look like color devices to QuarkXpress.

Paper Size/Width/Height

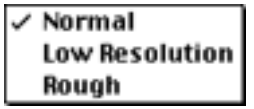
On occasion, QuarkXpress changes the paper size when you select a Distiller PPD file. Check and make sure these three controls reflect the page size you want for your final PPD file.

Next Page →

Options Tab In this set of controls, we need to pay attention to four controls:



Output Here you specify how QuarkXpress should print illustrations. You are given a choice among *Normal* and two draft qualities. Choose *Normal.*



Data Should image data be in ASCII or Binary? Choose *Binary.* Again, the normal reasons for picking ASCII (or the somewhat more efficient *Clean 8-bit)* don't apply if you're turning the PostScript file into PDF.

OPI Unless you are actually planning to print the final PDF file from an OPI server, you should select *Include Images.*

Full Resolution TIFF Output This should be turned on. It prevents QuarkXpress from downsampling images that it thinks have too much data.

Next Page →

That wasn't too bad

That's it. There are several controls requiring attention, but they're pretty straightforward in their meaning and use.

Other Applications

Other high-end applications (PageMaker, Corel Draw, etc.) give you controls similar to those in QuarkXpress. Look over their Print dialog boxes and set the ASCII/Binary and other controls as we've discussed here.

Microsoft Windows? Next Month.

Windows actually gives you much more control over the characteristics of your PostScript files. There's enough there that we'll put that off until next month.

Something to look forward to!

Case Statements in PostScript

If you write a lot of hand-written PostScript code, you occasionally find yourself having to decide among a number of alternative activities, determined by, say, the value of some variable. In C, this switching among possible tasks is best handled by a *case* statement: depending upon the value of *x,* do one of a number of things.

Unfortunately, PostScript doesn't implement a *case* equivalent, so we end up handling this with a set of nested *ifelse* statements, which looks very nasty in code:

```
x 5 eq
{ DoX5 }
{ x -3 eq
  { DoX-3 }
  { x -3 lt
    { DoXLessThan-3 }
    { DoDefault } ifelse
  } ifelse
} ifelse
```

This can be amazingly ugly to modify and maintain, especially if there are a large number of possible *x* values you want to trap.

It turns out that you can replace this with a very fast, very clean equivalent of *case* that uses dictionaries to hold the action procedures and to search. This is what we'll discuss this month: using dictionaries to do a *case* equivalent

Next Page →

Variable-Data Printing

Consider the variable-data printing world. These folks read data from a database and use PostScript code to generate quite elaborate documents unique to each set of data in the database. Typically, they will have a large PostScript header that defines procedures that do the work, followed by the invocation of some "DoItNow" routine, followed by newline-delimited data taken from the database.

In outline, the PostScript code looks like this:

```
Procedure definitions
...
DoTheReports
DataSet1
...
DataSet2
...
```

Usually, the first line of data in each set determines what data to expect for that set and what, exactly, to do with that data.

For our discussion this month, let us pretend that we are generating financial reports for client corporations. The first line of data in each set is the name of the company to whom the current report will be sent. Most companies get a generic report, but we want to special-case some company names so we can do something specific to them.

If the first line of data in each set is the name of the company, our PostScript code using *ifelse* would look something like this (stretches across two screens; sorry):

Next Page →

The Old Way...

```
/str 128 string def

% Company-specific procedures
/PottyProc      { (PottyProc) = } bind def
/MonstrousProc  { (MonstrousProc) = } bind def
/CrazyProc      { (Crazy) = } bind def
/DefaultProc    { (Default) = } bind def

/HandleCompany       % (CompanyName) => ---
{
     dup (Hot Doggie Portapotties) eq    % Hot Doggie?
     { PottyProc }                       % Yes
     { dup                               % Otherwise...
       (Monstrous Heights Real Estate) eq % Monstrous Heights?
       { MonstrousProc }                 % Yes
       {                                 % Otherwise...
          (Out of My Mind, Inc.) eq      % Out of My Mind?
         { CrazyProc }                   % Yes
         { DefaultProc }                 % Otherwise, do default proc
         ifelse
       }
       ifelse
     }
     ifelse
} bind def
```

Next Page →

The Old Way, cont'd

```
/DoReports
{
     {    currentfile //str readline  % Get company name from input stream
          not { pop exit } if         % Leave if no more companies
          HandleCompany               % Do something with this co.'s data
     } loop                           % Do it all again
} bind def

DoReports                            % Invoke the report procedure
Hot Doggie Portapotties              % Data.
Albequerque Turkey                   % Usually, each company name would be
Monstrous Heights Real Estate        % followed by data specific to that
Out of My Mind, Inc.                 % company.
Gorilla My Dreams Parties
```

Look long and hard at the nested *ifelse* statements on the previous page and imagine how very ugly this would look if there were, say, ten companies we wanted to special case. Adding additional companies would also be very trying and trouble-prone.

(By the way, all of the code for this month's article is available on the Acumen Training website's resources page: www.acumentraining.com/resources)

Next Page →

The New Way

So, let's fix things.

What we're going to do is create a dictionary to hold our company-specific procedures. Within this dictionary, the keys will be strings (not names) holding the company names as they will appear in the data stream. Associated with each company name will be the procedure that carries out company-specific activity. We'll also include a Default procedure for companies not in our special list.

In our new version of the sample program, we'll do the following:

- Read a company name from the input stream
- Check to see if that company name exists as a key in our dictionary
- If so, we'll execute that company's special procedure
- Otherwise, we'll execute the Default procedure.

The *known* Operator

Key to this will be the *known* operator:

```
<<dict>> key known => bool
```

This checks for the existence of a key in a given dictionary, returning a boolean *true* if the key exists, *false* otherwise.

Our new code looks like this:

Next Page →

The New Way...

```
/str 128 string def

/CompanyProcs        % This dictionary holds our company-specific procs
<<                   % Note the keys are strings, not names. This is OK.
     (Hot Doggie Portapotties)          { (PottyProc) = }
     (Monstrous Heights Real Estate)    { (MonstrousProc) = }
     (Out of My Mind, Inc.)             { (Crazy) = }
     /DefaultProc                       { (Default) = }
>> def


/HandleString        % (str) => ---
{
     //CompanyProcs exch       % Get the CompanyProcs dictionary
     2 copy known not          % If our company name isn't known...
     { pop /DefaultProc } if   % ...replace it with the name "DefaultProc"
     get exec                  % Get the procedure & execute it.
} bind def
```

Next Page →

The New Way, cont'd

```
/DoReports
{
     {    currentfile //str readline  % Get company name from input stream
          not { pop exit } if         % Leave if no more companies
          HandleCompany               % Do something with this co.'s data
     } loop                           % Do it all again
} bind def

DoReports                            % Invoke the report procedure
Hot Doggie Portapotties              % Data.
Albequerque Turkey                   % Usually, each company name would be
Monstrous Heights Real Estate        % followed by data specific to that
Out of My Mind, Inc.                 % company.
Gorilla My Dreams Parties
```

Comments

Notice that the *DoReports* procedure hasn't changed. It's still just reading the company names and handing them to *HandleCompany.*

HandleCompany is dramatically simpler. Four lines of PostScript code, as formatted here, and no *ifelse's* in sight. Adding a new company to the "Specials" list entails simply adding one more key-value pair to the *CompanyProcs* dictionary.

Furthermore, this executes much faster than the original *HandleCompany.* The comparison of each name with the list of specials is carried out as a hash search by the *known* operator, rather than a serial search in our PostScript code.

Next Page →

Dictionary keys and PostScript data types

One characteristic of this code that is significant in its implication is the fact that each key in the *CompanyProcs* dictionary is a string (except for */Default),* rather than the usual literal name.

Keys in a PostScript dictionary can be any kind of data: numbers, strings, executable names, anything. In our sample code, we switched off the company names, which were strings. We could have keyed off of zip codes:

```
/CompanyProcs
<<
     10012          { (PottyProc) = }
     92629          { (MonstrousProc) = }
     40263          { (Crazy) = }
     /DefaultProc   { (Default) = }
>> def
```

I even know one clever chap who created a dictionary that used operator *definitions* (not operator names) as the keys. He used this to selectively override PostScript operator definitions in *systemdict.* Very sophisticated use of dictionaries.

Next Page →

A PostScript *case* Statement

Using this technique, it is relatively easy to write a generalized *case* procedure. I'll leave this as a challenge for the readers; write a PostScript *case* procedure:

```
<< dict >>  obj  case  =>  ---
```

This procedure takes a dictionary and an object from the stack. If the object exists as a key in the dictionary, it executes the corresponding procedure. Otherwise, it executes a default procedure with some standard name. If a default procedure is not supplied, *case* should exit gracefully.

My version of *case* is included among the files for this month's Journal on the Acumen Training website. (Again, www.acumentraining.com/resources.)

Next Page →

Dictionary Underuse

Dictionaries are underused by most PostScript programmers. They create dictionaries for forms, images, and patterns; they'll put their own working dictionaries on the dictionary stack; and that's about it.

The fact is, dictionaries are amazingly versatile. In this article we used them for fast switching among alternative operations. But dictionaries can also be used where other languages would use structures, to hold a variety of data that should travel around together.

You can even do something very like object oriented programming in PostScript, using dictionaries as the basis for your objects. (Methods, properties, inheritance, overloading, you can pretty much do it all!) This would be another good exercise for the reader.

Or maybe a future Journal article.

Schedule of Classes, Apr 2001 - Jun 2001

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

The PostScript classes are taught in Orange County, California, near the Orange County airport, and in London at Adobe Systems' office near Heathrow.

PostScript Classes

PostScript Foundations	London, UK	May 7 - 11	Orange Co., CA	June 4 - 8
Advanced PostScript	Orange Co., CA	Apr 23 - 26	Orange Co., CA	July 16 - 19
PostScript for Support Engineers	Orange Co., CA	May 14 - 30	London, UK	June 18 - 22
Jaws Development	Orange Co., CA	Apr 30 - May 3		

For more classes, go to www.acumentraining.com/schedule.html

PostScript Course Fees PostScript classes cost $1,750 per student

Registration →
Acrobat Classes →

Acrobat Class Schedule

Acumen training teaches three users' classes in Adobe Acrobat (the links below will take you to the Acumen website's complete description):

Acrobat Essentials — This class teaches the student how to make perfect PDF files. It includes complete coverage of the meaning and proper settings of all of the Distiller Job Options.

Interactive Acrobat — Here we show you how to add bookmarks, links, buttons, sounds, movies, form fields, and other interactive features to an Acrobat file.

Troubleshooting with Enfocus' PitStop — This class shows the student how to use all of the capabilities of this popular editing and preflight software.

On-site Only — The Acrobat classes are taught only on corporate sites. If you have an interest in any of these classes for your group, please see the Acumen website regarding setting up an on-site class.

Back to PostScript Classes

Return to First Page

Contacting John Deubert at Acumen Training

For more information

For class descriptions or for any other information about Acumen's classes:

Web site: http://www.acumentraining.com

email: john@acumentraining.com

telephone: 949-248-1241

mail: 25142 Danalaurel, Dana Point, CA 92629

Registering for Classes

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

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

email: registration@acumentraining.com

telephone: 949-248-1241

mail: 25142 Danalaurel, Dana Point, CA 92629

Back issues

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

Return to First Page

What's New at Acumen Training?

U.K. Classes are Back

After a lengthy absence, the periodic PostScript classes are back at Adobe Systems' site near Heathrow airport. Acumen Training will be conducting PostScript classes several times a year beginning with a **PostScript Foundations** class this May and **PostScript for Support Engineers** in June. If you are in the United Kingdom or Europe, these classes may be much more convenient than coming to California. See the class schedule for upcoming classes.

New Class: *Jaws Development*

Acumen training has developed a new, four-day class in working with the *Jaws* PostScript interpreter. If you use this interpreter from Jaws Systems, this class will teach you how to write device drivers, device classes, PostScript operators, and to use all the other features of this versatile interpreter.

Requirements & Schedule

Students need to be proficient in C and should have reasonable knowledge of PostScript. (The PostScript Foundations class is highly recommended; you can better understand the *pagedevice* structure if you know what the PostScript *setpagedevice* does.)

The first scheduled class starts April 30. This class will be taught quarterly in Costa Mesa, California and, of course, on corporate sites. Go to the Acumen Training website for a course description.

Journal Feedback

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

Comments on usefulness. Does the Journal provide you with worthwhile information? Was it well written and understandable? Did you like it, hate it, or did it make you want to make faces at passers-by? How could we make it better? Do you like the PDF format?

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

Questions and Answers. We are planning a Q&A section for future issues. Do you have any questions about Acrobat, PDF or PostScript?

Please send any comments, questions, or problems to:

journal@acumentraining.com

Return to Menu