Table of Contents

Acumen Training

Making PostScript for PDF, Part 2

This month we continue our discussion of how to make PostScript files for conversion to PDF by Acrobat Distiller, PDF Creator or other conversion software. Last month we saw the controls available to us when making PostScript files on the Macintosh. Now let's see what controls Microsoft Windows offers us and how we should set up those controls.

Which Windows?

When discussing Microsoft Windows, of course, you need to specify exactly which flavor of Windows you mean: 95, 98, NT, 2000, ME, or what?

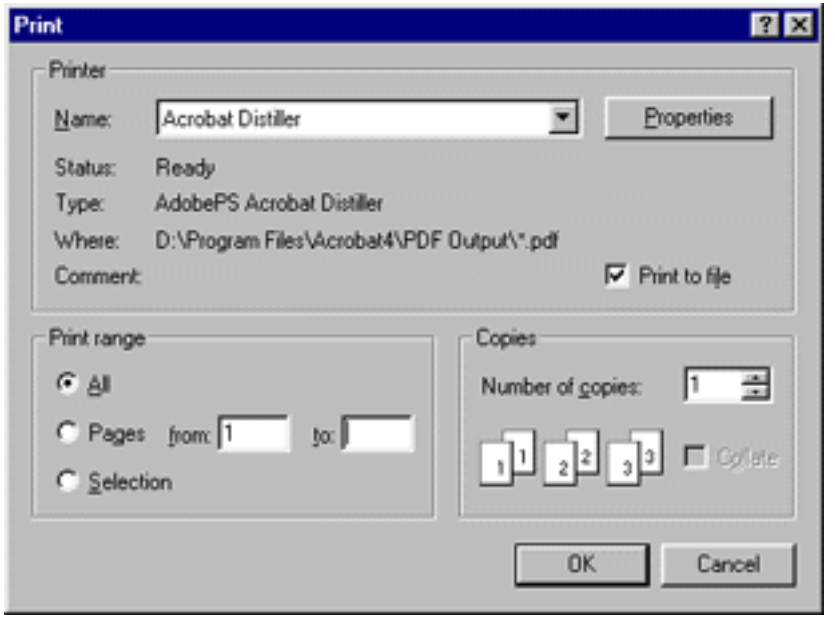
Actually, in our discussion we have it somewhat simple; the controls that affect PostScript are the similar across all versions of Windows. The dialog box layouts can be different, but the quantities you are controlling are the same: ASCII vs. binary, outlines vs bitmaps vs Type 42, etc.

In this issue of the Journal, I'm going to be using screen shots from the lowest common denominator: Windows 95. For other environments, you will need to take what I say here and look for the equivalent controls on your system.

Next Page ->

The Print Dialog Box

When you want to make a PostScript file for converting to PDF, the first step is to select "Print" in your application. You will be looking at the Windows Print dialog box.



There are two controls which you need attend every time you make a PostScript file:

Printer Name Here you specify the printer that Windows will assume when making a PostScript file. You want to select your target printer from the combo box's list of printers. In our case, we want to select our PostScript-to-PDF converter: *Acrobat Distiller,* etc.

Print to File: On

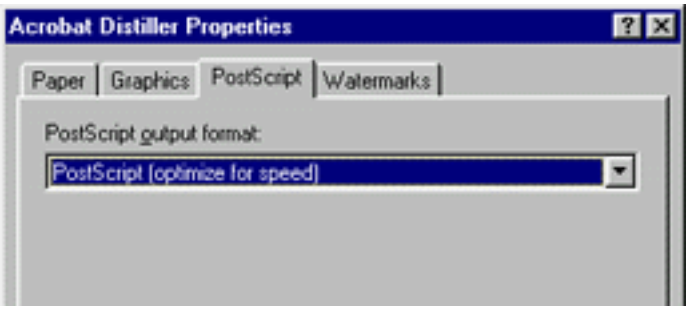
This check box tells Windows to create a PostScript file, rather than print directly to the printer. This should be selected, of course.

Next Page ->

Properties

If you click on the Properties button in the Print dialog box, you will be faced with a four-tabbed dialog box.

If you choose the PostScript tab, you will be presented with a single pop-up menu. This menu allows you to specify what kind of PostScript file you want to create. You have your choice among several different PostScript formats.



You want to choose **PostScript (optimize for speed)** when making a PostScript file for PDF. This will yield somewhat faster PostScript code.

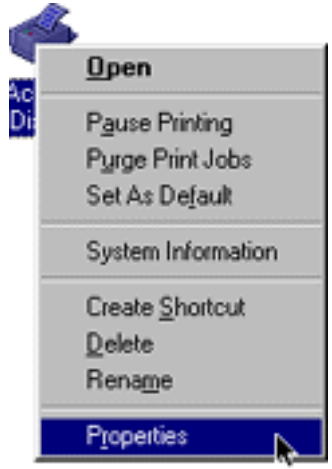
Note that this is different from the choice you would usually make if you were going to send this PostScript code directly to a service bureau. In that case, you would usually pick "optimize for portability." Since we are going to be converting our PostScript to PDF, the portability issue is much less important.

Next Page ->

Printer Properties

There are some additional controls that you should set up. To get to these controls, you need to go to the select Settings->Printers from Windows' *Start* menu. At this point, you will be looking at the Windows *Printers* window.

You want to right-click on the icon associated with your PostScript-to-PDF converter (Acrobat, etc.) and select *Properties.*



Windows will present you with a multi-tabbed *Printer Properties* dialog box. There are quite a few controls that are important here. Happily, these controls are all "sticky"; once you have set them correctly, you don't have to worry about them again.

Next Page ->

Fonts Panel

The first set of controls you should inspect in the *Printer Properties* dialog box is on the *Fonts* panel.

Here you have broad control over how TrueType fonts should be printed and, therefore, how they should be embedded in your final PDF file. There's only one control to pick here:

Always use TrueType fonts

Selecting this control ensures that the TrueType font you use in your layout is the font that makes it to the PDF file.

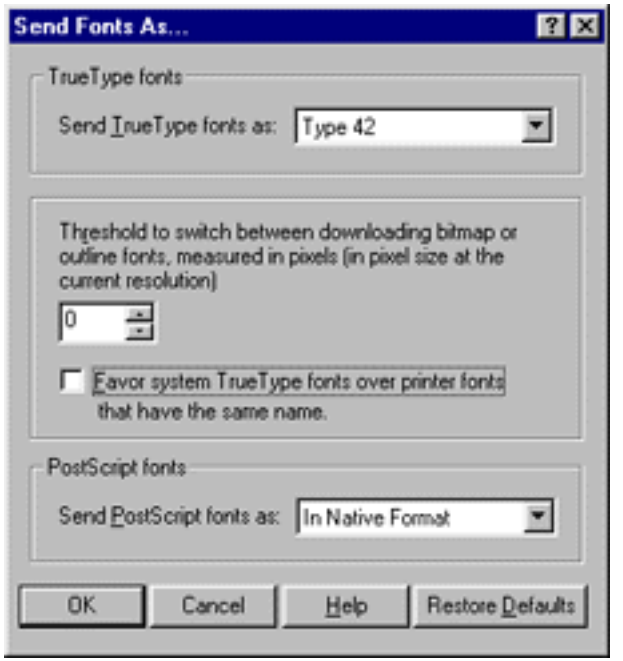
Having decided that we're going to embed our TrueType fonts in our PostScript (and, therefore, in our PDF file), we need to specify in what format those fonts should be embedded.

Click on the **Send Fonts As...** button.

Next Page ->

Send Fonts As...

Here we want to specify the format that should be used for embedding TrueType and Type 1 fonts. There are three controls you want to make sure are set correctly:



Send TrueType fonts as: Type 42

This will embed TrueType fonts in the final PDF file in their native format. Adobe Acrobat has no problem with native TrueType fonts and so there is no good reason to convert them to another format.

Favor System TrueType fonts...: On

Selecting this checkbox ensures that the fonts you used in the layout are the ones that are actually embedded in the PostScript and PDF files. (Otherwise, TrueType fonts that have a Type 1 equivalent in Distiller will not be embedded.)

Send PostScript fonts as: In Native Format

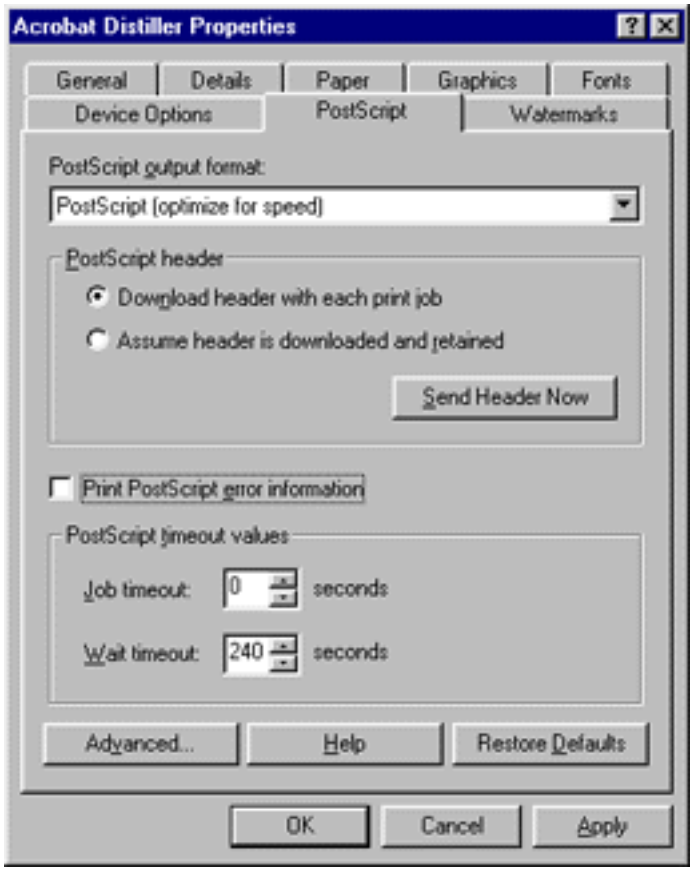
This tells Windows to embed Type 1 fonts in the PostScript file and your PDF file. Your only other choice is "Don't embed," which may cause your fonts to end up in the PDF file as lovely and attractive Courier.

Once you have set these appropriately, click on the *OK* button to return to the Printer Properties dialog box.

Next Page ->

PostScript Tab

The second tab you want to select in the Printer Properties dialog box is *PostScript.* Here we have access to controls that specify the details of the PostScript code generated by Windows.



Here we have only two controls to worry about, one of which we've seen already.

Format: PostScript (optimize for speed)
This is exactly the same pop-up menu we saw earlier. As before, we want to optimize our PostScript for speed.

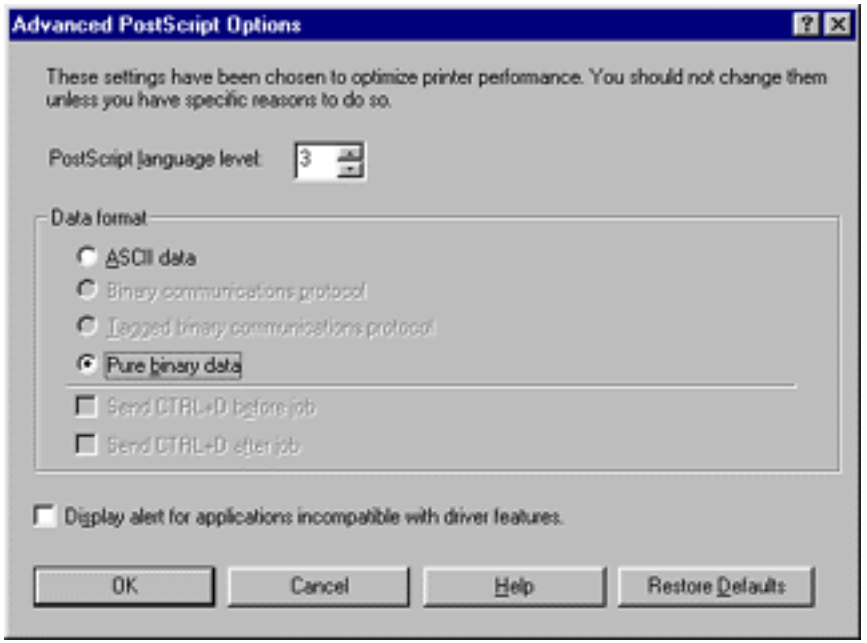
Download header with each print job
If you *don't* select this button, Windows will send only a partial PostScript file to disk; it will assume that Distiller has had the Windows PostScript header stored in its memory.

One last set of controls and we're done: click on the **Advanced...** button.

Next Page ->

Advanced PostScript Options

Finally, we have two controls among the advanced options that we must set correctly:



PostScript language level: 3

Acrobat Distiller, PDF Creator, and most of the current PostScript-to-PDF utilities are built around LanguageLevel 3 interpreters. Let's take advantage of this. The one cautionary note is that if you have a very old version of your conversion software (Acrobat 3 or PDF Creator 1.0), you may need to back this control down to LanguageLevel 2.

Data format: Pure binary data

ASCII format offers no benefits when you convert a PostScript file to PDF. You should use the more compact binary format.

Click the **OK** button until you are back in Windows. You're done.

Next Page ->

That's It

All you have to do now is return to your application and print to a PostScript file. Keep in mind that every time you print to PostScript, you should check the printer pop-up menu and the Print dialog box's Properties to make sure they are set correctly.

You need to set the Printer Properties only once.

Return to Main Menu

Making *eexec*-Encrypted PostScript Files

If you've been in the PostScript biz for a while, it has probably happened to you that you come up with a piece of PostScript that was hard to write and that does something important to your application. You'd just as soon not give your hard work away to the world at large.

Unfortunately, PostScript code is awfully difficult to keep secret. Just about anyone can open it with a text editor and see how you solved your problem.

There is a solution to this built into PostScript: the *eexec* operator. This operator can execute the contents of a file or string that contains encrypted PostScript code. The encryption scheme has long been published, so this doesn't make the PostScript truly secure. It does, however, hide your code from casual visitors.

This month... This month, *PostScript Tech* will review how the *eexec* operator works and then take a look at an ANSI C program that converts a PostScript file into an encrypted PostScript file.

The C code for this month's example is available on the Acumen Training website at www.acumentraining.com/resources.html.

Next Page ->

The *eexec* Operator

The *eexec* operator was introduced into the PostScript language for the use of the Type 1 font mechanism. Type 1 fonts start out in clear PostScript that ends with a call to *eexec.* The encrypted PostScript that follows *eexec* makes up the great majority of the font definition.

Adobe published the encryption algorithm when they published the Type 1 font spec, so the *eexec* encryption has been an open secret since 1990 or so.

The *eexec* operator takes a file object or string object from the operand stack and executes the contents of the file or string.

```
fileobj eexec => ---
(str)   eexec => ---
```

The contents of the string or file must be properly encrypted PostScript in either binary or hexadecimal format.

Next Page ->

Mixing Encrypted and Unencrypted Code

If *eexec's* argument is a file object, the operator will execute the contents of that file until end-of-file or until the file is closed. Thus, if you want to have only part of a PostScript file be encrypted, reverting afterward to plain PostScript, you must either:

- Execute the file through the *SubFileDecode* filter or other filter that allows you to explicitly specify an end-of-data. Your invocation of *eexec* would look like this:

```
currentfile 0 (*ENDOFEEXEC*) /SubFileDecode filter eexec
...encrypted PostScript goes here...
*ENDOFEEXEC*
... back to normal PostScript ...
```

- End the encrypted PostScript by closing currentfile. That is, the final PostScript execution in the encrypted code should be:

```
currentfile closefile
```

  This works because *eexec* places its source file, with an attached filter, onto the Execution stack. The *closefile* operator will cause this encrypted file object to be removed from the Execution stack and the interpreter will revert to unencrypted execution.

Next Page ->

Acumen Journal: PostScript Tech

eexec and the Dict Stack

The *eexec* operator places *systemdict* on top of the dictionary stack before executing the encrypted code. As a consequence, all PostScript operators in the encrypted PostScript will execute with their default, native-PostScript definitions.

Also as a consequence, if the encrypted PostScript wants to define any key-value pairs, it will need to put its own dictionary on top of the dictionary stack. Otherwise, the *def* operator will try putting the definition into *systemdict,* which is read-only.

Next Page ->

The Encryption Algorithm

The *eexec* encryption algorithm is published in the *Adobe Type 1 Font Format* manual, available from Adobe's website: partners.adobe.com/asn/developer/pdfs/tn/t1_spec.pdf.

To convert clear PostScript to encrypted PostScript:

1. Generate four arbitrary bytes to be added to the beginning of the clear PS. If you are going to be generating hexadecimal code, then all four of these must be hexadecimal numerals ("A" - "F," "a" - "f," "0" - "9"). Conversely, if you are generating binary encrypted code, at least one of these bytes must *not* be a hexadecimal numeral.

2. Initialize an unsigned 16-bit integer, *R,* to the encryption key 55665.

3. For each byte, *P,* of plain PostScript code (beginning with the four bytes we prepended), do the following:

   - Exclusive-OR the plaintext byte with the high order 8 bits of *R;* the result will be the encrypted byte value, *C.*

   - Calculate the next value of *R* as follows:

     $$R_{next} = ((C + R) \times C_1 + C_2) \bmod 65536$$

     where

     $$C_1 = 52845 \qquad C_2 = 22719$$

Note that each byte's value of *R* is calculated from the preceding byte.

Next Page ->

Acumen Journal: PostScript Tech

Doing it in C

The ANSI C code that implements this algorithm is pretty straightforward. To make things easy, Adobe supplies C code to do this a byte at a time. My code below is a bit different, mostly in that it converts the PostScript a buffer at a time.

The C code converts the contents of a PostScript file into an encrypted PostScript file that starts with a call to *eexec.* The listing here has been edited for brevity; there are parts missing. (For example, I've dropped all the procedure declarations and *#include*'s.)

The full program is on the Acumen Training resources web page: www.acumentraining.com/resources.html. Look among the PostScript sample code.

Note that this program is written for clarity, not necessarily as excellent final code. (For example, I've hardwired constants with the names of the source and destination files and have dropped any pretense at error trapping.) Feel free to fix this up as appropriate to your needs.

This program stretches across three page, even with parts left out.

Next Page ->

The C Code

Some constants

```
#define kSrcFileName  "source.ps"
#define kDestFileName "dest.ps"
#define kLogFileName  "source.log"
#define kBufferLength 4096
```

Next Page ->>

main()

```
int main(void)
{
     FILE            *src;                  // This is our original PostScript file
     FILE            *dest;                 // This will receive our encrypted PS
     unsigned char   *buffer;
     size_t          count;
     char            phonyBytes[] = "WXYZ";    // Our initial four bytes

     src = fopen(kSrcFileName, "rb");          // Open our files
     dest = fopen(kDestFileName, "wb");
     buffer = (unsigned char *)malloc(kBufferLength); // Allocate our buffer

     fputs("%!PS\rcurrentfile eexec\r", dest); // Write to dest our call to eexec

     EncryptBuffer((unsigned char *)phonyBytes,4); // Encrypt our phony bytes…
     fwrite(phonyBytes, 1, 4, dest);               // …and write them to dest

     while (!feof(src))   {                        // For the entire source file:
          count = fread(buffer, 1, kBufferLength, src); // Read PS code…
          EncryptBuffer(buffer, count);                 // …encrypt it…
          fwrite(buffer, 1, count, dest);               // …& write it to dest.
     }

     fclose(src);
     fclose(dest);
     free(buffer);

     return 0;
}
```

Next Page ->

Acumen Journal: PostScript Tech

EncryptBuffer()

```
void EncryptBuffer(unsigned char *buffer, size_t len)
{
	// Here are our encryption values
	static unsigned short	r = 55665;
	static unsigned short	c1 = 52845;
	static unsigned short	c2 = 22719;
	unsigned char		*b, *b0;

	b0 = buffer + len;
	for (b = buffer; b < b0; b++) {
		*b = (*b ^ (r >> 8));
		r = (*b + r) * c1 + c2;
	}
}
```

The Result

When executed, the program produces a file containing a call to *eexec* followed by the encrypted PostScript. In the case of a test run here at Acumen Mansion, the contents of the destination file were:

```
%!PS
currentfile eexec
ÑÖH"0o;W5¸Î □Pyêáî¯!õ'{∫ bƒ Ëå˜¯'˜'ŒR dã¡üîÚ 0G†[ë"â§˜˝1∑+Z%º{‰:÷—)
```

Next Page ->

Code on the Website

The code on the Acumen Training website differs from the above in that:

- It's complete. Compile it and run it and it will convert a PostScript file to an encrypted PostScript file.
- It does some minimal error trapping. If it can't open one of the files or if *malloc* fails, the program exits gracefully.

Feel free to use the code on the website as a starting point for a more elaborate program of your own. You might want to generate encrypted output in hexadecimal format, implement an interface (command line, if nothing else) that lets you specify the source and destination files, etc.

It is rather fun to hide your PostScript from at least casually prying eyes.

Return to First Page

Schedule of Classes, May 2001 - July 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	Orange Co., CA	June 4 - 8	Orange Co., CA	August 6 - 10
Advanced PostScript	Orange Co., CA	July 16 - 19		
PostScript for Support Engineers	Orange Co., CA London, UK	May 14 - 30 June 18 - 22	Orange Co., CA	July 23 -27
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. The Jaws class is $1,900 per

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 Training website regarding arranging 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?

Acrobat Classes Move to 5.0

Both the Acrobat Essentials and Interactive Acrobat classes have been upgraded to cover Acrobat 5.0. The new version of Acrobat presents us with a new set of additional Job Option controls that one needs to set correctly to make successful PDF files. Additionally, Acrobat 5.0 offers new features that open new possibilities. The Acrobat classes now include these new features and options.

Jaws Development Rolls out

The first-ever *Jaws Development* class is being held April 30 - May 3. If you are a Jaws OEM, you owe it to yourself to look into this class. From basic invocation to writing device drivers and classes, this class provides a solid foundation for using the Jaws PostScript interpreter with your printing device or application.

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 next scheduled class is July 30 - August 2 in Costa Mesa, California. It can be also be conducted, of course, on corporate sites. Go to the Acumen Training website for a course description.

Return to First Page

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 eat brussels sprouts? 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