# PostScript Startup Secrets

*Don Lancaster*

***Most PC users go far out of their way to blindfold themselves and tie both their feet together when they first try to apply PostScript. By understanding a few simple first setup principles and some insider user secrets, PostScript can easily become as PC-friendly as it is on all of the other machines.***

For openers, please do not ever call PostScript a page description language. To do so is a gross and demeaning insult that's pretty much comparable to referring to UNIX as a "checkbook balancer".

Instead, PostScript is one totally *general purpose* computer language which can hold its own against any modern contender. While it happens to excel in dirtying up any otherwise clean sheets of paper, you'll find that page descriptions and page layouts form only an inconsequential corner of PostScript's real abilities.

As a general purpose computer language, PostScript often ends up strongly object-oriented, especially in its EPS file transfer modes. There are usually *three* different forms of PostScript output, allowing you to make markings upon sheets of thin stuff; to return useful information for host recording, external control, and user interaction; or to write directly to local or shared SCSI disk files.

PostScript is rather highly *device independent*. On the input end, this means that any old text file from any old editor or word processor on any old computer can be used as your program source. On the output end, the very same PostScript file can be used to drive anything from a low resolution desktop laser printer up through the fanciest of high precision phototypesetters. PostScript can also be used for embroidery, sign cutting, desktop prototyping, plotting, video editing, engraving, CAD/CAM, FAX interchange, image compression, or any of a host of other emerging new applications.

PostScript is related to Forth, as a second cousin three times removed and five times disowned. As such, PostScript uses the *postfix* notation (reverse Polish having no goto's); is *extensible* (you can define a nearly infinite variety of new operators as combinations of all existing ones); is *reentrant*, (any operator or proc can call any other one to any reasonable depth); is *stack oriented* (providing for inherent self-addressing); and is *loosely typed* (you do not have to rigidly predefine everything).

PostScript is exceptionally good at manipulating strings and arrays. It is strongly dictionary oriented, giving you local and global constructs. In a mind-boggling feature not available in any other common language, your dictionary entries can relate *any* two of the dozens of object types, not just key-value pairs.

PostScript does its linear graphical transformations on the fly. Thus it is simple for you to translate, rotate, scale, or otherwise transform any image to any size or any position at any time. PostScript has built in cubic spline capabilities that use *Bezier Curves*. These can easily let you draw smooth flowing curves given only a sparse set of the ending and control points. Your use of these splines lets you create typography by using a procedural description rather than with a bitmap. Only a single outline description (combined with suitable hinting rules) is needed for all sizes and all variations of any user-chosen font. Unlike the *hershey bar effect* of scaled bitmaps, scaled splines will get *smoother* as they are enlarged.

Sophisticated font machinery is an inherent part of PostScript. A two-step process is used. The first time a character is needed, it gets built up from a tightly compacted description. At the same time, a bitmap copy of the final character is usually saved to a *font cache*. Repeated reuse of the same character then can get imaged much faster.

The latest PostScript *Level II* adds some surprising new features, such as a full FAX capability, instant and unrestricted access to all font paths, higher speeds, improved color seps, strong forms and step-and-repeat capabilities, JPEG video compression, and bunches more.

As a general purpose computer language, PostScript is fast and fun to use. You could generate award

winning output with nothing but five minutes of instruction and a dozen simple commands. Or you can spend a lifetime exploring and pushing the envelope of an incredibly powerful new multimedia communications tool. But be forwarned: PostScript is extremely addictive.

You do not normally go out and buy your own copy of the PostScript language. Instead, the language is built into or gets added on to most popular laser printers. It is important that you realize that any PostScript speaking printer is really a general purpose computer which executes PostScript programs. An extremely powerful computer with a fast and an exceptionally clean architecture that is lots of fun to program and use. It is definitely *not* a "just dump the text" printer. And does not become one until such time as you or some firmware teaches it exactly what you want it to become.

### Programming in PostScript

A PostScript program is usually nothing more than a plain old ASCII textfile that contains ordinary words and numbers. PostScript programs can be *preinstalled* as firmware or cartridges; can be *downloaded* on the fly; or can be *persistently downloaded* once to remain available as long as computer power remains turned on. Downloading can be done via serial ports, AppleTalk, shared SCSI comm, Ethernet, or directly to or from a modem, hard disk, or CD ROM.

As we will see shortly, one-way parallel ports should definitely *not* be used with PostScript under any circumstances. Ever.

There are several popular ways of generating PostScript code. One is to use *raw PostScript* to write your own programs using your favorite word processor or editor. This one is my overwhelming favorite and is by far your most powerful and the most flexible. Raw PostScript also usually generates by far the shortest and the fastest-running files as well.

A second method of providing PostScript code is to use an *emulator*. An emulator is a currently running PostScript program which accepts an input in some other format and converts it on the fly into suitable PostScript commands. The popular traditional emulators include PCL, Diablo, Imagewriter, Epson, HPGL, and TEX. Most users usually outgrow traditional emulators twelve minutes or less into their use. ["Gee. I'd sure like some larger drop titles, an initial cap, proportional spreadsheet fonts, and a box around that sidebar..."]

PostScript Level II adds powerful new *filter* emulators. Some of these early offerings include full FAX and JPEG video compression.

The third method of generating PostScript program code is to use an *applications package* to write the code for you. Obvious examples include *Ragemaker*, *Adobe Illustrator*, *Ventura Publisher*, and *Aldus Freehand*.

Although fast and convenient for beginners, all of these application packages force you to do their own things in their own way and generate code that is inherently slower, lots longer, and far less flexible than by writing your own raw PostScript.

*EPS files* are related to PostScript code generators. These are simply pretested and (hopefully) debugged textfile program modules that have a standard header and trailer. These also disallow certain obscure PostScript operators that can cause havoc when they get imported into another supervisory program. EPS is short for *Encapsulated PostScript*.

### Setting Up Your Serial Comm

All PostScript programs *demand* two-way host recordable and fully on-screen displayable comm.

**Any communication, networking, or print buffering setup that prevents you from immediately receiving on-screen error messages or from your recording any host-returned data is severely crippling you and will surely**

**▲ FIGURE 1 – A PostScript two-way communications verifier.**

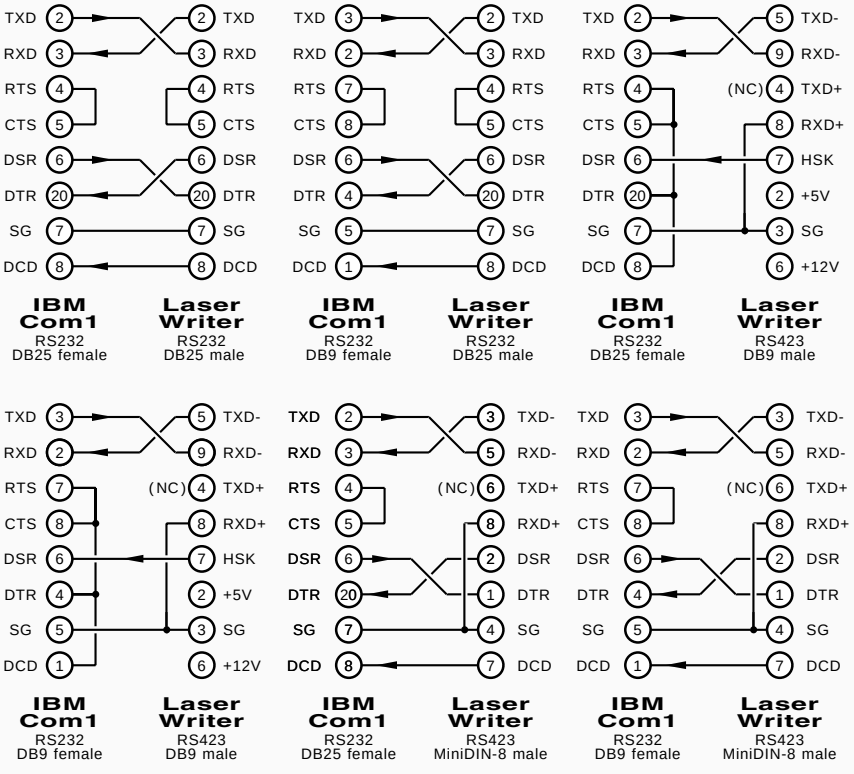
PostScript **DEMANDS** full two-way host comm, especially for on-screen error messages and for host recording of returned data.

Make **ABSOLUTELY CERTAIN** you can run this 13-byte PostScript program…

**(OK) == flush**

If your comm setup, network, or print buffer does not display the OK on your host screen, then **STOP IMMEDIATELY** and make your system PostScript usable.

**▲ FIGURE 2 – PC to LaserWriter cable options.**



**▲ FIGURE 3 – PostScript serial comm initial checkout.**

1. Connect the PostScript printer to a suitable **serial** cable, select a 9600 comm mode, apply power, and verify the baud rate on the test page.
2. Boot a suitable **two-way** comm program such as *Crosstalk* or *ProComm*. Select 9600 baud, 8 data bits, 1 stop bit, no parity, full duplex, all filters off, and XON/XOFF handshaking.
3. Type a single **ctrl-T**. A message of…

   **%%[status: idle]%%**

   should appear on your host screen. If nothing happens, your cables are not properly crossed or your comm is not properly activated. If gibberish appears, your baud rate or other comm parameters are wrong. If a different legible PostScript message appears, reset the printer per the next step or otherwise attend to the problem (e.g. add paper; replace cartridge; free a jam, etc.)
4. Reset your printer by typing a **ctrl-C** followed by a **ctrl-T** half a second or more later. You once again should get the **%%[status: idle]%%** message promptly appearing on your host screen.

   **DO NOT GO BEYOND THIS POINT WITHOUT YOUR PROPER ERROR MESSAGES BEING CORRECTLY DISPLAYED ON SCREEN!**
5. Reset your printer again and type the word **showpage**, one word and all lower case. A page should eject. Reset your printer again and type your middle name. An error message should appear on-screen. Reset your printer again and run Figure one. An **OK** should appear on your host screen.
6. To run other PostScript code, first create and save the code to disk as a standard ASCII textfile. Use your favorite editor or word processor. Then transmit the code with your comm program. Try figure four as a useful example.
7. Note that improper XON/XOFF handshaking will not affect short programs, typically those under 5000 characters or a page and a half. Do **NOT**, under any circumstances, attempt to use DTR handshaking.

**return to haunt you.**

Virtually all the non-IBM users of PostScript have a *SendPS* or similar program that *always* provides them with a full and instant two-way recordable host comm.

Figure one shows you a thirteen byte PostScript program to verify that you have a useful comm setup. If you fail to get an "OK" on your host computer screen, then you should **immediately** get rid of your present comm setup and replace it with a decent one. The longer you wait to do this, the more it will cost you in time, money, and frustration.

**Any one-way parallel port should NEVER be used with PostScript!**

And those printer manufacturers that lead you to believe otherwise are doing a severe disservice to both you and the PC community at large.

These epsilon minuses should get staked to their nearest anthill. Or chopped up and fed to the cows. Or, at the very least, spanked and sent to bed without any supper.

Many PostScript programs (such as figure one, that *Adobe Distillery*, the GENIEVST.PS stock investment analysis code, or countless others) will not run at all on a one-way port. And those programs that do run will place you under an extreme comm disadvantage. Even for such trivial matters as running out of paper or going low on toner.

On the other hand, there is a great programming opportunity here. For the first *PC Techniques* reader to come up with an equivalant to *SendPS* that provides a full two-way recordable comm over a parallel port is certain to run away with a very large bag of marbles. But, until that happens, you definitely must **not** use any one-way parallel port with PostScript.

Yes, you can find printable error trappers. These can greatly simplify program testing and debug. And you certainly should. Our *GEnie* PSRT #196 EHANDLER.PS is one source. But by no stretch of the imagination is a printing error trapper a suitable substitute for fully recordable two-way comm and instant on-screen error messages.

By the same token, COPYing any PostScript program to your serial COM1 port is one big no-no. Again, because it flat out prevents you from receiving all your timely and crucial on-screen error messages and from host recording returned data. And also because it usually prevents the normal PostScript XON/XOFF handshaking protocol.

### Serial Cables

Figure two shows you six popular serial cables, useful for connecting Apple LaserWriters to PC or clone computers. On the baseline DB25 to DB25 connector, the usual mistake people make is forgetting to locally jumper the auxiliary RTS and CTS pins (4 & 5) on the IBM end. These serial pins are an IBM anachronism not found elsewhere.

Note that the data and optional handshake paths are crossed in what you IBM folks call a *null modem*, and what Apple calls a *printer* cable.

The usual mistake when you are connecting RS232 to the RS423 used by DB9 or MiniDIN-8 connectors is failing to ground the "unused" RXD+ input. Be sure to use the connections shown. RXD is differential.

Note especially that older Apple LaserWriters make optional use of a RS423 DB9 connector whose pinouts end up wildly different than the DB9 RS232 used on newer IBM or PC gear. Neither a "straight" nor a "crossed" DB9 cable can be used. Worse, it is likely to cause fireworks and possible damage at both ends.

Rather than building up a special cable, it is usually simpler to take any old cable and an ohmmeter and find out what pins you really have. Then you go to *Radio Shack* and pick up a DB-25 male and a female connector and some short jumper wires. You then build an adaptor for one end of your cable that makes the cable you have into the one you really want. Ready-to-jumper modules are also readily available.

### Your First Comm Test

A good first comm debug routine is shown you in figure three.

Connect up the proper cable and select the correct baud rate on your PostScript computer. (That's that box that wrongly has a "printer" label on it.) Then boot a suitable full *two-way* comm program such as *Crosstalk, PoComm* or whatever you are using with your modem. Initially, select 9600 baud, eight data bits, one stop

bit, no parity, full duplex, filters off, and XON/XOFF handshaking.

Then enter a **ctrl-T**. If all is well, you should get an on-screen message of **[status: idle]**.

*Always* reset your printer before any reuse. To do this, you do a **ctrl-C** followed by a **ctrl-T** half a second or more later. There is *never* any point in continuing further if you cannot get the **[status:idle]** message.

Next, enter the word **showpage**, as one word, all lower case. A blank page should promptly eject out of the PostScript computer. Reset once again. Now type your middle name. You should get an on-screen error message that reminds you that you forgot to previously define your middle name as a PostScript proc.

The general procedure for running a raw PostScript or an EPS routine is to save all your code as a standard ASCII textfile to disk. Then get into your favorite comm program. Type a [C][T] to get the [status:idle] message. Then send your file. Any problems should immediately appear on your host screen.

Figure four is a practice PostScript program. It does a fractal fern, and can be stunningly beautiful. For your first few passes, keep your *numdots* variable in the thousands. Then raise it as high as you want for a detailed final hard copy.

By the way, this fern program is an excellent PostScript clone buster. It works just fine with genuine Adobe PostScript but totally demolishes most of the PS imitators, fakes, and hanger-onners.

It is usually a good idea to defeat your comm program local echoing. Having to take time out to display outgoing characters and to scroll screens dramatically slows down all your de-facto baud rates. Especially at higher settings.

After your comm seems up and working, you can next check out all of the PostScript drivers on your applications software to verify that they are working properly. *There is no point in trying anything with a fancy package before you verify that your basic two-way recordable comm is up and working with raw PostScript.*

You can later speed your comm up to 19200 or even 59600 baud. Details vary with the printer, but you can call me per the end blurb for more help on this.

**▲ FIGURE 4 – Use this PostScript fractal fern as a practice file.**

```
/problistcreate {mark /counter 0 def probabilities {128 mul round cvi {transforms counter get}
repeat /counter counter 1 add def} forall counttomark 128 sub neg dup 0 gt { [1 0 0 1 0 0] repeat}
{pop} ifelse] /problist exch def} bind def

/doit {problistcreate 1 1 20 {problist rand -24 bitshift get transform 2 copy moveto .001 10 rlineto}
repeat newpath numdots {problist rand -24 bitshift get transform 2 copy moveto .001 0 rlineto
stroke} repeat} bind def

%%  /// demo - remove or alter before reuse. ///

/numdots 6000 def   % increase for denser image; decrease to print faster

/transforms [ [0 0 0 .16 0 0] [.2 .23 -.26 .22 0 1.6] [-.15 .26 .28 .24 0 .44]
[.85 -.04 .04 .85 0 1.6] ] def

/probabilities [ .01 .07 .07 .85] def

1 setlinecap 0 setlinewidth 200 300 translate 30 dup scale doit showpage quit
```

Note that any PostScript printer setting marked 9600 is often fully reprogrammable. Should you royally foul things up, your setting marked 1200 is usually always 1200 baud, 8 data bits, and 1 stop bit. You could always drop back to this setting and then reprogram the other one.

It is normally a good idea to start and end all of your files with a [D] end-of-file command.

Obvious ways to divert a single serial port for several uses include manually replugging cables, using an A-B switch box, or substituting a third-party card that has multiple serial ports.

### Transparent Data Channels

Any communications setup has a big dilemma. There has to be some means to control the channel you are communicating over. Do you add extra control wires or reserve some characters for commands?

PostScript normally uses a nearly (but not quite) transparent *scc* data channel. The reserved characters are shown you in figure five.

As long as you send only printable ASCII characters, you will never have any transparency problems. The real transparency killer is that [D]. Any inadvertent hex $04 or "CHR$(04)" causes an end-of-file termination of your PostScript program. With any XON/XOFF handshaking, the [Q] and [S] characters are also obviously reserved and must never be used for any other purpose.

If you decide to use any control characters in your raw PostScript code or emulator, you have to make absolutely certain they don't conflict with the uses of figure five.

How can you handle your images, "perfect" emulators and binary data if some codes are restricted? There are several popular ways. One is to use hex-ASCII pairs. Each data value is broken down into a high byte and a low byte and separately sent as printable ASCII 0-9 or A-F characters. This always will work and is totally transparent, but requires twice the characters and transmits at half the usual comm speed.

An improvement on hex pairs that got built into Level II is known as ASCII85. This is basically a base 85 numbering scheme that substitutes *five* printable ASCII characters for *four* unrestricted binary bytes. The speed and storage penalty for a full

**▲ FIG. 5 - Reserved Characters.**

| | |
|---|---|
| **ctrl-C** | Force job reset |
| **ctrl-D** | end-of-job marker |
| **ctrl-J** | Newline or linefeed |
| **ctrl-M** | Carriage Return |
| **ctrl-T** | Inquire printer status |
| **ctrl-Q** | XOFF stop handshake |
| **ctrl-S** | XON restart handshake |
| **ctrl-H** | Interactive backspace |
| **ctrl-R** | Interactive redisplay |
| **ctrl-U** | Interactive erase line |

transparency is only 20 percent.

Yes, you could defeat PostScript's reserved characters by creative use of the undocumented *openscc* operator in *statusdict*. But this is a tricky and dangerous process. The proc format is *9600 25 0 openscc*.

### [J] 's and Such

PostScript will do its darndest to remain totally device independent. But there are just enough differences between the Mac and the IBM comm standards that you do have to watch out. Changes may be requied when moving any supposedly "device independent" textfile from one system to the other.

For instance, the Mac normally uses only [M] carriage returns, while IBM often uses [M][J] carriage return and linefeed pairs. An IBM textfile will often double space on a Mac. In PostScript code, this often will not matter, but within PostScript strings or *currentfile* readings, results may end up double spaced. Similarly, a Mac textfile can appear as one single humongous long string on an IBM screen or disk file.

The obvious solution here is to add or remove the [J] linefeeds during a transfer. Many utility programs give you an option to do just this.

Another problem character is [Z]. Many IBM file transfer programs will fill out the end of their listing with enough [Z] characters to come up to an even page boundary. These same [Z] characters can give PostScript or a Mac fits if they are not properly predefined. It pays to check the end of your files for extra characters, and then strip off any unneeded ones.

Certain older versions of the Mac AppleTalk reception code in some printers wrongly failed to convert the [M] carriage returns into the [J] PostScript newline characters. This can cause imaging differences with some PostScript text scanner codings. Rewrite your text scanning code if your AppleTalk images differently than serial comm.

A final possible problem character is the tab command [I]. Some systems substitute tabs for multiple spaces and vice versa. Any [I] routed to a PostScript program which does not expect it can generate errors.

Some word processors may insist on forcing extra unwanted carriage returns, especially at a screen line ending or when importing or fully formatting text. It is super important to prevent any *unintended* carriage return introductions when creating or sending PostScript text. Especially if those carriage returns can end up inside PostScript strings or are present during *currentfile* readings.

### Type 1 Fonts

In their essential form, Adobe Type I *printer* fonts are simple text files that start out with a readable header and then drop into a compact *eexec* encoded file. Any of these files in their "bare bones" format can be downloaded device independently from any host source.

Machine specific extensions are sometimes added to Type I fonts. For instance, a Mac Type I font will have a *data fork* and a *resource fork*, and may include some pre-ATM *QuickDraw* screen character bitmap info. Thus, a Mac Type I font can't simply be moved over and used on a PC.

There are several simple ways to convert Mac Type I fonts into a PC usable form.

One method requires a PostScript printer with a local hard disk. Just Mac download the Type I font to disk. Then read the font back to a PC host. Note that the only difference between any old hard disk textfile and a font is that the font is placed in a */fonts/* subdirectory.

A second method needs a Mac application swift enough to "go fish" for host based fonts not already on the printer. Create a short textfile that needs the target font. Then print, but choose either an *option-F* [print ASCII file to disk] or the *option-K* [print ASCII file and all the needed persistent dictionaries to your disk]. Transfer the disk-based text file to a PC and then strip off everything that is not the Type I font itself.

A third method uses a shareware program to automatically convert the font formats for you. The *GEnie* PSRT #207 UNADOBE.SIT is typical.

Going from PC to the Mac is far simpler. Make sure your file contains only your Type I font printer info. Then use the *Download PostScript File* option of *SendPS* to download your font to the printer.

Be sure to note that their *Download PostScript Font* option will expect the Mac format font listing, while their *Download PostScript File* selection accepts an ordinary ASCII text file from any source.

### For More Help

Two essential PostScript books are the Adobe *blue* book, and otherwise known as the *PostScript Tutorial and Cookbook*, and the revised *red* book, which is titled the *PostScript Reference Manual II*. The red book includes full details on EPS files, Level II, and on new *Display PostScript*.

LaserWriter tech info appears in the Apple *white* book, also called the *LaserWriter Technical Reference*.

I am also now laboring under the delusion that my *Intro to PostScript* video and my new *LaserWriter Secrets* book/disk combo might be of help to you. I carry lots of other PostScript products by Adobe, by myself, and by other leading authors. There is a no-charge PostScript voice helpline here at *Synergetics* which you can call per the blurb below.

Finally, a major and PostScript-oriented BBS is up on *GEnie* as PSRT. Included are hundreds of demos, utilities, fonts, and my power tools, along with RTC conferences with all major PostScript players.

You can voice call (800) 638-9636 for connect info. Your typical downloading charges for an average PSRT file is around twenty-one cents. ❖

---

*Microcomputer pioneer and guru Don Lancaster is the author of 26 books and countless articles. Don maintains a no-charge technical helpline found at (602) 428-4073, besides offering all his own books, reprints, and his technical services. He also has a free brochure of his insider desktop publishing secrets waiting for you. Your best calling times are often 8-5 on weekdays, Mountain Standard Time. Or you can reach Don by way of* Synergetics, *at Box 809, Thatcher, AZ 85552.*