# Using Adobe's Acrobat Distiller as a General Purpose PostScript Computer

**Don Lancaster**
**Synergetics, Box 809, Thatcher, AZ 85552**
**copyright c2003 as GuruGram #29**
**http://www.tinaja.com**
**don@tinaja.com**
**(928) 428-4073**

One of our most important and most popular **Guru's Lair** downloads has been **DISTLANG.HTML** As this was in an older format and from way back in the days when **Adobe Acrobat PDF** web access was rare, it seemed to be sorely in need of a **Sherwin-Williams** overhaul. Which I thought we might do here.

The key point being…

> **The Adobe Acrobat Distiller can easily be used as a superb general use host based PostScript interpreting computer!**

And a surprisingly fast and amazing powerful one. The **PostScript** language itself can be both incredibly versatile and a lot of fun to use. Why, it is even rumored that **PostScript** may be capable of dirtying up otherwise clean sheets of paper! But surely few people would limit themselves to such a restrictive use.

I have already applied it for everything from **robotics controls**, making **bitmap image corrections**, reporting **eBay Image thefts**, doing hassle free **PowerPoint emulations**, working with **energy efficiency breakthroughs**, handling **digital camera swings & tilts**, automatically **writing complex JavaScript programs**, and, of course, for pretty near all of my general engineering design.

I've also found **PostScript** to be exceptionally handy in its ability to **read or write virtually any diskfile** in pretty near **any** format. Thus using **elegant simplicity** to easily solve great heaping bunches of otherwise ugly compatibility problems.

Much of what you will need to get started can be found on our **PostScript** and **Acrobat** library pages. Especially our **PostScript Beginner Stuff** series. Additional help is available directly from **Adobe Systems** as well as the **comp.text.pdf** and the **comp.lang.postscript** newsgroups.

Normally, you (or an application) creates a **PostScript** program in the form of a **standard ASCII textfile**.

You will send this program to **Acrobat Distiller** to create an **Acrobat PDF** file for distribution. The usual essence of distilling is try and get rid of everything **not** specifically needed to make final marks on a page.

Thus, to be able to properly distill, the program must be able to handle any and all **PostScript** thrown at it.

Distiller normally generates **two** documents: A **.PDF** file usable for later viewing or printing of graphic output. Plus a **.LOG** file containing any **print** or **==** results from the input program, along with status messages. Distiller can optionally also be used to create **disk based** output files. Once written, these files can be used for port control or **any** computational purpose whatsoever. In **any** language!

As a simple example, say you want to find the sine of 35.4 degrees. Create the following program in **WordPad** or another editor or word processor…

```
%!
35.4 sin ==
```

Save your file as an ordinary text file with a **.ps** trailer. Drag and drop this file in the **Adobe Acrobat Distiller**. The correct answer of **0.579281** should quickly pop up in the dialog box. A later reading of the **.LOG** file will return this value, along with an expected warning message that no actual **.PDF** file is produced.

The rules are that…

**PRINTING or == commands go to the .LOG file as text.**

**MARKING commands such as show, fill, stroke, or showpage go to the .PDF file as imaging actions.**

**WRITING or Reading commands work with your own custom defined output or input disk files.**

There are lots of times when using **PostScript** as language that you throw away your **.PDF** baby and drink the **.LOG** file washwater instead.

Two more key rules…

**Always start your program with the %! character pair!**

**Do NOT end your program with a quit or a control [d]!**

The **.LOG** file can be a quick and dirty way to get useful **PostScript** output.

Logfile use does have some quirks you should be aware of. Ferinstance, **Distiller** performs a **flushfile** on every newline character. Nicely letting you see problems immediately as they occur.

But any **PostScript** program of yours that includes lots of print commands that include newline characters or **flush**es or **==** will **(a)** slow things down, and **(b)** cause a lot of disk clatter.

Thus, writing a special disk based output file can end up faster, cleaner, and less demanding any time that many newline characters are involved.

A second **.LOG** file hassle: **Long strings are truncated to 200 characters!** The rest of the string gets replaced by the three ellipsis dots  Workarounds are to use shorter strings, characters in an array, or your own special output file. Fortunately, long arrays or procs do get properly reported.

Other nasty habits of log files are that **your end log result will be a mix of your print commands and processing status and error messages**. On an error, **the log file stack dump is also truncated to 200 entries.**

## Reading and Writing Disk Files

Some of the most versatile and powerful PostScript-as-language stuff you can do involves reading and writing disk files. These can arrive in virtually any language and be written in virtually any language. Thus, **PostScript** is superb for **oddball file conversions** or **resolving format conflicts**.

PostScript disk commands are discussed in depth in **PSDISK03.PDF**. Here are a few summary examples…

**To CREATE a PS disk file object for READING…**

```
(C:\\WINDOWS\\Desktop\\sourceinfo.txt) cvn (r) file
/infile exch store
```

**To READ one line of an INPUT disk file…**

```
/workstring 20000 string store
{infile workstring readline {dostuff}{exit} ifelse} loop
```

**To CREATE a PS disk file object for WRITING…**

```
(C:\\WINDOWS\\Desktop\\targinfo.txt) cvn (w+) file
/outfile exch store
```

**To WRITE one line of an OUTPUT disk file…**

```
outfile (string_to_be_written) writestring
```

Note that the PostScript **(w)** command starts a new file each time, while the **(w+)** **appends** an already existing file. Fancier reading and writing procs also exist for single bytes or random access.

More rules…

> **ALWAYS use "\\" whenever you really want a "\" in any of your PostScript filename strings!**
>
> **ALWAYS use the "long form" FULL pathname with Windows.**

It is usually best to always create **new** disk files, rather than overwriting or modifying existing ones. It is also a good idea to **read or write most files a full line at a time** rather than character by character. This will often be much faster and far less demanding on the disk drive itself.

**PostScript** also lets you use readable and writable strings as virtual file objects. Through their **NullEncode** and **SubFileDecode** filters. Subject to a **65K** limit.

## The Gonzo Utilities

I long ago wrote a set of PostScript **Gonzo Utilities**. These are a **non-WYSIWYG** combination of pagemaking and illustration tools that do such tasks as really great text justification, premium quality electronic schematics, and superb grid-based graphic presentations. Plus many other PostScript extensions.

Full details on these appear on the **Gonzo** panel of our **PostScript** library. Typical detailed use examples appear in most any of our **.PSL** sourcecode files. Such as those found on our **GuruGram**, **Tech Musings**, or **Blatant Opportunist** libraries.

To add these capabilities to your own PostScript programs…

> **MAKE a copy of GONZO.PS available on your hard drive.**
>
> **ADD this line near the start of your PostScript program…**
>
> **(C:\\windows\\desktop\\gonzo\\gonzo.ps) run**
>
> **ACTIVATE the utilities where first needed with a…**
>
> **gonzo begin  ps.util.1 begin  nuisance begin**

The best way to get started with Gonzo is to work on through our **PostScript Beginner Stuff** projects. These give you bunches of things like letterheads, business cards, advertising aides, menus, bumperstickers, badges, and great heaping bunches more to explore on your own.

## Pathforall Oddities

Distiller provides a rather bizarre twist on the **pathforall** operator. Pathforall will operate normally on any font properly installed into ATM. It will also operate uselessly but as expected on a **Courier** substitution for a non-Adobe or with any mis-spelled font.

But pathforall will return a **bounding box** for any requested but uninstalled Adobe font. Watch this strange detail very closely.

## Essential PostScript Resources

Two "must have" documents are the **PostScript II Reference Manual** and the **PostScript III Reference Manual**. These are downloadable free using these links, or may be purchased in hard copy via **Amazon Books**.

The more interesting and more obscure info sometimes appears in the PostScript Language supplements…

**PostScript Language Supplement 2011**
**PostScript Language Supplement 2012**
**PostScript Language Supplement 2013**
**PostScript Language Supplement 2014**
**PostScript Language Supplement 2015**
**PostScript Language Supplement 2016**

**PostScript Language Supplements 3011 and 3012**

Additional PostScript support appears on this **Adobe PostScript Technotes** site, on my **PostScript Libray** and **Acrobat Library** pages, my **PostScript Disk Access Notes** utility file and by way of the **comp.lang.postscript** or the **comp.text.pdf** usenet newsgroups.

A collection of third party links can also be found **here** and **here** .

## PostScript Limitations

If **PostScript** is so great, why isn't it more widely used as a general purpose computing language? Firstoff, because few people yet realize how easy, how convenient, how fast, and how powerful it has recently become.

Second, because **PostScript** appeals largely to **bare metal** types who can fully appreciate the elegance and strength of a non-mainstream **stack oriented** language. And, finally, because **Adobe** lately been de-emphasizing general PostScript-as-language use due to extensions in the Acrobat imaging model.

Some other perceived PostScript restrictions may include…

**Full Acrobat is Required —** Distiller is available on all commercial and educational versions of Acrobat, but is **not** included on any of the free Acrobat readers.

**PostScript is Proprietary —** Source code for your own custom apps is generally **not** available, except for outrageously high fees. However, there is a free to low cost **GhostScript** variant that **does** offer fully customizable sourcecode.

**Raw PostScript is "not quite" WYSIWYG —** If you make a change in a raw PostScript program, it may take a second or two and a few mouse clicks to view the actual results. But once again, that **GhostScript** alternative offers immediate viewability. But at the price of being a tad klutzy and somewhat behind the curve.

**Raw PostScript works best in a batch mode —** In which you give it all the needed info ahead of time, let it do its thing, and then observe or view the final results.

**I/O is restricted —** Raw PostScript is normally limited to inputs of internal instructions and readable disk files. Its output is similarly restricted to **.PDF** files, log files, and written disk files. However, this can be expanded with **Acrobat Plugins** or **GhostScript**.

**Interactivity is limited —** It is difficult for raw PostScript to get a user input mid-program. But, once again, this can be handled through the use of **Acrobat Plugins** or **GhostScript**. PDF **form** submissions are also a possibility.

**Scripting is external —** There's no obvious provision in PostScript for automatic or unattended external control. Many tasks are best done "semiautomatically" where each job is user specified. But external scriptors can be added on a custom basis. The Acrobat **watched folders** feature may also be of help here.

**Any interpreted language runs slower than a compiled one —** But the instant convenience and freedom from all front end prep hassles often can **reduce** your total job time.

**Poorly written PostScript runs slow —** Most tightly written raw PostScript distills blindingly fast on any modern host. In general, your own code can be rewritten 3X to 10X faster than its first cut, and outside code can often be sped up beyond 10X to 100X by using **these techniques** .

**Newest PDF features go beyond the PostScript model —** The use of alpha **transparency** remains tricky with raw PostScript. As may other future **Acrobat** capabilities yet to be introduced.

**Math accuracy is limited —** PostScript uses 32-bit math, roughly equal to six or seven decimal places. While good enough for everyday use, **precision scientific calculations** may need the 64-bit accuracy of **JavaScript** or another language.

Math accuracy reporting can be improved by using **these techniques**.

Needless to say, I've never let any of these restrictions get in my way at all. For **PostScript** remains one of the most versatile and most fun to use general purpose computer languages of all time.

## An Important Update

Starting with Acrobat 8.1, the ability to read and write disk files **has been disabled** as a default. To use our Gonzo utilities, this read and write ability **must** be restored.

In Windows, the Distiller file read and write ability is restored by using a command line run command of…

```
acrodist -F
```

The read and write enabling will remain active so long as an executable version of Distiller is on or under the desktop. To use, you simply drag and drop your Gonzo routines into this active instance of Distiller.

## For More Help

A full **Gonzo** tutorial newly appears **here**.

Additional **PostScript** and **Acrobat** and additional assistance is available per the previously shown web links. Custom programming and design services are now available at our standard consulting rates. Per our **InfoPack Services**. Or you can directly **email** me.

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