Some PostScript Disk Access Notes

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552

http://www.tinaja.com
don@tinaja.com
(928) 428-4073

The general purpose PostScript computing language from Adobe Systems has a number of different disk access features. These include support of high end printers and typesetters that have internal hard drives for font, page, job, and form storage. And the ability for Acrobat Distiller to access and modify files on its own host operating system.

In expanding upon an InfoPack I did for a client, I thought I'd gather together and review some of the more obscure PostScript disk access fundamentals.

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 disk access info usually 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 earlier PostScript Disk File Utilities utility file and by way of either the comp.lang.postscript or the comp.text.pdf usenet newsgroups.

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

Snooping Around

The PostScript language is fairly open and inspectable, so it is often a good idea to snoop around inside to find the available resources. But note that…

> You must have two way comm to return any useful info
> from a PostScript peripheral, printer, or imagesetter!

Many early PostScript peripherals were conventional parallel port driven and thus unable to return any info or messages. To intelligently work with PostScript these days, you must use both two-way comm and software capable of receiving and interpreting returned info.

Suitable two-way comm methods include serial, USB, SCSI, Ethernet, AppleTalk, new IEEE high speed bidirectional parallel, or IDE. To make sure you have two-way comm, just send your middle name to your PostScript device. You should get back an undefined error message or log file.

The host based Acrobat Distiller, of course, eliminates any comm hassles and lets you directly run interpreted PostScript without comm worries. As found in this Using Acrobat Distiller as a General Purpose PostScript Computer tutorial.

Your essential starting point on any PostScript snoop is the version command. This should tell you whether you are working with PostScript Level I, II, or II, with GhostScript, or with a third tier clone. A "real" level two PS device should return something like 2014, while a level three will use a fancier 3010.157.

Once you know your version, you can dump and save both your systemdict and statusdict dictionaries„

```
statusdict  begin  statusdict  {exch == == (\n) print } forall

systemdict begin systemdict {exch == == (\n) print } forall
```

There are bunches of other dictionaries which you should be able to "avalanche" out of systemdict. One "magical" dictionary is internaldict which does need an easily found password and special treatment…

```
1183615869 internaldict begin 1183615869 internaldict
{exch == == (\n) print } forall
```

One of the more interesting commands buried in internaldict is superexec, which sometimes lets you access restricted info…

```
1183615869 internaldict begin
{restricted stuff {access proc} forall} superexec
```

IODevices and File Systems

In general, there are two main types of disk files associated with PostScript or Acrobat access.

Files internal to an imagesetter or a high end printer will often make use of the Adobe File System format and consist or an UNIX-like collection of 1024 byte pages. The first disk is typically called (%disk0%), but oddball names are sometimes substituted. A (%*) can often be used as an "any disk" wildcard. As we'll shortly see, these files are easily created, read, written, and cataloged.

Adobe Acrobat instead accesses whatever the (%os%) host operating system happens to be using. While you can usually create, read, and write files to the host system, I have been unable to find any means of cataloging host system files with Acrobat. Nor of finding used and available disk space. Nor of using relative filenames. Apparently these restrictions may have been purposely done to reduce virus and other malicious file potential.

Typically, PostScript or Acrobat will link IODevices of one sort or another. Some of these will be FileSystem devices that can have files created, read, written, or deleted. FileSystem devices can sometimes be initialized, sized, or cataloged.

With PostScript III, you can list your IODevices with...

```
/str 50 string def
(*){==} str /IODevice resourceforall
```

Doing so may return this list for Distiller...

```
(%hostfont%)
(%Distillery%)
(%os%)
(%fontset%)
(%cid%)
(%ram%)
```

The properties of each of these IODevices can be found by using...

```
(%hostfont%)currentdevparams {exch 50 string cvs print
(    ) print == } forall (\n) print
```

... which should tell us that (%Distillery%) is a Communications IODevice, while all the others are FileSystem IODevices. Writeable and readable disks would be included as FileSystem devices. Similar code can be used to find the available Disk Systems for most any PostScript printer or imagesetter.

Level II Disk Commands

There are only six PostScript level II disk commands of primary interest. They may apply only to Adobe File System format devices. These have been retained in level III as compatibility operators...

devstatus — *string* devstatus *searchable writeable hasNames mounted removable searchOrder free logicalSize true (if device known) false (if not)*

Returns the values of various file system attributes for the disk device identified by string (which must be %diskn% for some integer value of n). If the specified device name is known, the boolean value true is returned on the top of the stack, accompanied by the values of the device parameters Searchable, Writeable, HasNames, Mounted, Removable, SearchOrder, Free, and LogicalSize If the device name is unknown, only the value false is returned on the stack.

Errors: stackoverflow, stackunderflow, typecheck

diskonline — diskonline *bool* (statusdict dictionary)

Returns a boolean value indicating whether there exists a writeable disk device. The result will be true if there exists a device parameter set named %diskn% (for some integer value of n) in which the value of the Writeable parameter is true; if no such parameter set exists, a false value will be returned. Disk does not need an initialized file system.

Errors: stackoverflow

diskstatus — diskstatus *free total* (statusdict dictionary)

Returns the current number of free blocks and the total number of blocks available on all writeable disk devices. These values are determined by finding all known device parameter sets named %diskn% (for integer values of n) in which the value of the Writeable parameter is true, and totaling the values of their Free and LogicalSize parameters, respectively.

Errors: stackoverflow

initializedisk — *blocks action* initializedisk (statusdict dictionary)

Initializes each writeable disk by setting the device parameters LogicalSize and InitializeAction in each %diskn% parameter set to the values blocks and action + 1, respectively. This operator should be invoked only from within a system administrator job.

Errors: invalidaccess, ioerror, rangecheck, stackunderflow, typecheck

setuserdiskpercent *int* setuserdiskpercent (statusdict dictionary)

Formerly set the percentage of disk space reserved for user files. This operator now pops the operand int off the stack and otherwise does nothing. This operator should be invoked only from within a system administrator job. There is no Language Level 2 equivalent for this operator.

Errors: invalidaccess, rangecheck, stackunderflow, typecheck

userdiskpercent userdiskpercent *int* (statusdict dictionary)

Formerly returned the percentage of disk space reserved for user files. This operator now does nothing and returns the value 0. There is no LanguageLevel 2 equivalent for this operator.

Errors: stackoverflow

Level III Disk Parameters

PostScript Level III instead uses resource dictionaries that can be expanded for future capabilities. In the listings that follow, read-only refers only to access by language operators (for example, setdevparams and currentdevparams).

The value of a read-only parameter can change, but not as a result of invoking setdevparams. Changes to parameters in FileSystem parameter sets take effect immediately.

Device parameters in the %diskn% parameter set:

Type *name* (Read-only)	The parameter set type; must be FileSystem.
Bus *string* (Read-only)	The name of the bus on which this disk resides %Scsi% (or) if a SCSI bus, %Ide% (or %IdeX%) if an IDE bus. If the imaging system uses a file system of the native operating system rather than the Adobe storage device implementation, this parameter may not be meaningful or may be absent. More info about the bus (for this disk device) can be obtained by using currentdevparams.
HasNames *boolean* (Read-only)	A flag specifying whether the device supports named files. Since a device will not mount successfully unless it contains a valid file system, HasNamesis always true for mounted devices; if false, the device is not mounted.

Mounted *boolean*

A flag specifying whether the device should be mounted or dismounted, or (if queried) whether the device is currently mounted. Setting Mounted to true indicates that the system should attempt to mount the device, and setting it to false that it should attempt to dismount the device.

Mounting a device makes it known to the system and makes it at least readable, depending on the nature of the device. A device will not mount successfully unless it contains a valid file system (that is, unless the HasName parameter is true). If an attempted mount (or dismount) fails, a configurationerror occurs.

When queried, the value of this parameter indicates whether the device is currently mounted. Mounted should be queried immediately after it is set, to obtain the attempted mount or dismount result.

Removable *boolean* (Read-only)

A flag specifying whether the device supports removable media. Depending on how the removable media device operates, setting the value of Mounted to false for such a device will either eject the medium or allow the medium to be removed. Once the medium has been removed, the device cannot be mounted until the medium is reinserted.

Writeable *boolean* (Read-only except during a mount)

A flag specifying whether the files on the device can be opened for write access (or, if the device is not mounted, whether the device will support writeable media). If the medium is writeprotected, Writeable is false. This parameter can be set only during a mount- that is, at the same time that the Mounted parameter is being set to read only if the medium is not write-protected.

Searchable *boolean*	A flag specifying whether the device participates in searches for a file when a file name is supplied that does not specify a device (see PLR3, Section 3.8.2). Note: On some products, devices that support removable media will initially have Searchable set to false. Searchable must be explicitly set to true on such devices to enable them to participate in device searches.
SearchOrder *integer*	The priority (0 or greater) at which the device participates in file searches as indicated by a value of true for the Searchable parameter. The lower the value, the higher the priority. This parameter is ignored if Searchable is false.
BlockSize *integer* (Read-only)	Number of bytes in a block on the formatted device. For a disk using the Adobe file system, BlockSize is 1024. This parameter determines the unit for the values of the Free, PhysicalSize, and LogicalSize parameters.
Free *integer* (Read-only)	Number of blocks of free space available on the medium in the device (where the block size is indicated by the BlockSize parameter). Free is 0 if the medium is completely full or if the device is not mounted.
PhysicalSize *integer* (Read-only)	The number of blocks of medium in the device (where the block size is indicated by the BlockSize parameter). This is the maximum allowable size of the file system; the exact size is determined by the LogicalSize. PhysicalSize is 0 if the device is not mounted.
LogicalSize *integer*	The number of blocks allocated to the file system (where the block size is indicated by the BlockSize value). When queried, LogicalSize is 0 if the device is not mounted. When set, this parameter specifies the number of blocks to be allocated to the file system when it is created in response to the InitializeAction parameter.

The value of LogicalSize must not exceed that of PhysicalSize. If LogicalSize is 0, then InitializeAction uses a default size that is normally the same as the value of PhysicalSize.

If LogicalSize is set with a certain value and (as specified by the value of InitializeAction), the medium in the device is reformatted before the file system is created, a subsequent query of LogicalSize should return the value that was set. However, if LogicalSize is queried before the medium is reformatted, it may return the current size instead.

InitializeAction *integer*

A code specifying an action for initializing the device:

0 — No action. This value is returned when the parameter is queried.

1 — Deletes current file system. (if any) and creates one having the size specified by the LogicalSize parameter. (Medium is assumed to have been formatted already.) The device must first be mounted; otherwise, an ioerror will occur.

2 — Reformats entire medium before creating a new file system of size LogicalSize. The Interleave parameter also plays a role in how the medium is formatted.

3+ — Same effect as the value 2 and also carries out product dependent actions, which typically consist of preformatting the disk and running integrity tests before creating the file system. Some devices can have additional parameters that serve as arguments to InitializeAction.

Interleave *integer*

The interleave number, n, specifying n-to-1 interleaving. Interleaving arranges logically contiguous sectors on the disk in the most efficient way for the system using that disk. This parameter is used only when the medium is being formatted by InitializeAction.

For example, assume there are 16 sectors going around a single track on a disk. If the first sector is logical sector number 1, the second 2, the third 3, etc... the value of Interleave is 1 (1-to-1 interleaving). In this case, the system must be very fast to be able to take data from the disk, one sector immediately after another. If the system fails to consume the first sector in time for the second sector, it has to wait an entire revolution of the disk to get the next sector. This can result in very poor performance.

If the first sector is logical sector number 1, the third 2, the fifth 3, and so on, the system needs to consume the current sector while the head skips over a sector in time for the next logical sector. In this case, the value of Interleave is 2 (2-to-1 interleaving). The sectors in between are used for higher logical numbers. It takes a minimum of two revolutions to get the data for an entire track off the disk. In this example, the second physical sector on the disk would be between logical sectors 1 and 2, and would be logical sector 9.

Similarly, with an Interleave value of 3, the first sector is logical sector number 1, the fourth 2, and so on.

Normally, Interleave gets set to a value that lets the software use the data during the time between sectors, but not waste any time. It is difficult to determine what the proper value is. The value depends greatly on the job accessing the disk. For drives that provide buffering for a full track of data, 1-to-1 interleaving is almost always most efficient.

PrepareAction *integer*

An action to prepare the underlying file system for a specific purpose:

0 — No action.

0 — Device specific action.

Some Examples

Here are a few simple examples of PostScript disk access routines. In a fully blown program, you'll want to add such bells and whistles as a conditional known command to prevent errors should the commands not be available.

But first, an important rule...

> Always use a "\\" double reverse slash inside a filename or other PostScript string every time you really want a "\" or single reverse slash!

Things can really get complicated in Gonzo where you'll need four reverse slashes for every one that is to actually appear on your screen or page.

Merge two strings... This routine is excerpted from my PostScript Gonzo Utilities that are found on the PostScript Library page. It is quite handy when manipulating filenames and such. The two top stack strings become one...

```
/mergestr {2 copy length exch length add string dup dup
4 3 roll 4 index length exch putinterval 3 1 roll exch
0 exch putinterval} def
```

Check available AFS disk space — Should work for most PostScript printers and imagesetters using the Adobe File System format...

```
statusdict begin
diskonline dup == flush true {diskstatus == == flush} if
end
```

This (and similar routines) may or may not work for Distiller (%os%) disk operating systems, depending on the presence or absence of PhysicalSize and Free parameters. Possible workarounds may be available through the Acrobat SDK Software Development Kit.

Catalog AFS disk — Disks are normally named (%disk1%), (%disk2%), etc... . A (%*) can usually be used as a wildcard...

```
/str 50 string def
(%*){== } str filenameforall
```

On Distiller using (%os%) disk operating systems, this code may sometimes only return a fontlist found in (%hostfont%).

Write to an AFS disk using currentfile — Rename and prepend the following to any text file to be stored on a laser printer's hard disk...

```
/filename (filenamehere) def
filename status {4 {pop} repeat filename deletefile} if
/mfn filename (w) file def
/buffer 1024 string def
/bf {{currentfile buffer readstring pop dup length 0 eq
   {pop mfn closefile exit}{mfn exch writestring}ifelse
   } loop}def
bf
```

Once again, double reverse slashes should be used in any PostScript string where a single reverse slash is wanted.

Prepare file for Distiller host disk read or write — This also shows how to use our mergestr utility...

```
/diskfileheader (C:\\WINDOWS\\Desktop\\inherit) store
/diskfilesourcename (g9demoyx.pdf) store
/diskfiletargetname (g9demozz.pdf) store
/sourcefilename diskfileheader diskfilesourcename
      mergestr store
/targetfilename diskfileheader diskfiletargetname
      mergestr store
/readfile sourcefilename  (r) file store
/writefile targetfilename (w+) file store
```

Note that the (w+) tells us to append file info, rather than overwriting it. You can use similar code to read or write AFS files by changing to the (%disk1%) filename scheme.

Read a file one byte at a time — After you predefine a /yourreadfilename to either AFS or (%os%) naming conventions...

```
/strx (X) def
/yourreadfilename (r) file /myworkfile exch def
{myworkfile read {strx exch 0 exch put
your_character_user_routine_goes_here }
{myworkfile closefile exit} ifelse
} loop
```

The your_character_user_routine must consume a one character stack-top string in some manner.

Read a file one line at a time — After you predefine a /yourreadfilename to either AFS or (%os%) naming conventions...

```
/strx 256 string def
yourreadfilename (r) file /myworkfile exch def
{myworkfile strx readline
  {your_string_user_routine_goes_here}
  {myworkfile closefile exit} ifelse
} loop
```

The your_character_user_routine must consume a one line stack-top string in some manner. /strx can be lengthened or shortened as needed. It must be larger than the longest line in the file.

Change the position of the next file read or write — After myworkfile is predefined and in use...

```
/nextfileloc 12345  def
myworkfile nextfileloc setfileposition
```

This scheme lets you randomly access file info. Rather than sequentially.

Strings as "Disk" Files

PostScript strings can be made to appear as read or write files by applying suitable filters to them. This lets you use many of the disk file commands for string manipulation. The only major restriction is that your strings will have the usual 65,536 character limit.

The two magic filters are...

NullEncode — *source* NullEncode *filter*

Passes all data through, without any modification. This permits an arbitrary data target (procedure or string) to be treated as an output file.

SubFileDecode — *source count string* SubFileDecode filter

Passes all data through, without any modification. This permits an arbitrary data source (procedure or string) to be treated as an input file. Optionally, this filter detects an end-of-data marker in the source data stream, treating the preceding data as a subfile.

source is the original string. countis the maximum number of characters to be delivered. string is an end of file marker, such as a carriage return or tab character.

Here's a sneaky example that shows you the power of strings-as-files...

Convert an array to a string — A stacktop array of 8-bit integers can get converted to a string using this code...

```
/makestring {dup length string dup /NullEncode filter
3 -1 roll {1 index exch write} forall pop} def
```

While file techniques are certainly not needed here, their Elegant Simplicity makes for a very cute routine.

Our PostScript Fractal Fern gives you a second example of string-as-file manipulations. Per this FERN2IMG.PSL sourcecode. The same utility can be used to convert most any calculation-intensive PostScript routine into a fixed size bitmap image. Giving you a much faster print or view time at the expense of fixed size and resolution. Here's the calculated fern for a speedup comparison.

More PostScript utilities appear on our PostScript Library page. Additional string-as-file resources appear in Adobe Tech Note TN5603

Some More Detailed Applicatons

Here's a list of some of our other PostScript disk manipulation files...

pdf2bmp.psl	Acrobat Bitmap Typewriter
acatdata.psl	Acrobat Catalog Internal Data Formats
catwords.psl	Acrobat Catalog Word List Extractor
graburls.psl	Acrobat URL Extractor & Link Tester
blender.psl	Bitmap blender and manipulator
catools1.psl	Catalog Data Manipulation Tools
strconv.html	Convert PS Strings, Integers & Arrays
bodcat.psl	Duplex Catalog Auto-Addresser
flatvue1.psl	Flate Compression Object File Viewer
fern2img.psl	Fractal Fern to Image Converter
flutools.psl	Flutterwumper Utilities
mscal156q.psl	Fourier Equation Generator (plus more in the Magic Sinewave library)
weblogu2.psl	Log File Interpreter Utilities
pfa2pfb.psl	PFA to PFB Font File Converter
pfb2pfa.psl	PFB to PFA Font File Converter
psdisk03.psl	PostScript Disk Access Notes
disktool.psl	PostScript Disk Tools
psinscrt.psl	PostScript Insider Secrets
pssearch.psl	PostScript Search & Replace

more...

refsum1.html	Referral Log Analyzer
reflog1.html	Referral Log Reader
reflog1.psl	Referral Log Report Generator
refsum1.psl	Referral Log Summary Analyzer
searepl.psl	Search & Replace Demo
weberru2.psl	Web Error File Utilities
grabsrch.psl	Web Query Log Extractor
grabrefs.psl	Web Referral Log Extractor

For Further Help

Consulting assistance on any and all of these topics can be found at http://www.tinaja.com/info01.asp.

Additional GuruGrams await your ongoing support as a Synergetics Partner.