Adobe Developers Association

ADA news

%!PS-Adobe-3.0

You may have seen many PostScript language files that begin with `%!` or `%!PS-Adobe-3.0`. Perhaps your application begins each of its PostScript language jobs with one of these comments … but do you really know what these comments mean?

`%!PS-Adobe-3.0` is the first line in a file that conforms to the document structuring conventions (DSC), version 3.0. This article gives an overview of the DSC, and explains why it is crucial for PostScript printer drivers and other PostScript language generators to create output that conforms to the DSC. In addition to this article, please refer to technical note #5001, "Document Structuring Conventions, version 3.0" or Appendix G of the *PostScript Language Reference Manual, Second Edition.*

What is the DSC?

The DSC is a standard set of document structuring conventions. The conventions consist of both a set of special comments which communicate the structure, requirements and contents of a PostScript language file, as well as a set of guidelines and restrictions for the actual PostScript language code in the file.

DSC = comments + structure and code requirements

DSC Comments

The first line of a DSC-conforming file is `%!PS-Adobe-x.y`, where *x.y* is the version number of the DSC specification to which the document adheres. Non-DSC comments begin with a single percent character (`%`), and can appear anywhere on a line of code. All DSC comments, with the exception of the initial `%!` comment and the Open conventions, are specified by two percent characters (`%%`) as the first characters on a line.

Note: A non-conforming document is recognized by the `%!` header comment. Under no circumstances should a non-conforming document use the `%!PS-Adobe-3.0` header comment. There is no such thing as a "partially-conforming document". A document either conforms to the DSC or it does not.

There are various DSC comments used to specify document structure and the following content information: General, Requirements, Color Separation, Query, Open and Special. This article addresses the General, Requirements and Color Separation conventions categories. For details about all of the convention comments, see the DSC specification.

General Conventions—impart basic information about a PostScript language page description, such as its bounding box, page orientation, and title. General convention comments also delimit the structural components of a document, including its prolog, script, trailer, and page

continued on page 2

How To Reach Us

DEVELOPERS ASSOCIATION HOTLINE:

U.S. and Canada:
(415) 961-4111
M–F, 8 a.m.–5 p.m., PDT.
If all engineers are unavailable, please leave a detailed message with your developer number, name, and telephone number, and we will get back to you within 24 hours.

Europe:
+31-20-6511-355

FAX:

U.S. and Canada:
(415) 967-9231
Attention:
Adobe Developers Association

Europe:
+31-20-6511-313
Attention:
Adobe Developers Association

EMAIL:

U.S.
devsup-person@mv.us.adobe.com

Europe:
eurosupport@adobe.com

MAIL:

U.S. and Canada:
Adobe Developers Association
Adobe Systems Incorporated
1585 Charleston Road
P.O. Box 7900
Mt. View, CA 94039-7900

Europe:
Adobe Developers Association
Europlaza
Hoogoorddreef 54a
1101 BE Amsterdam Z.O.
The Netherlands

Send all inquiries, letters and address changes to the appropriate address above.

breaks. For example, the beginning of a page description in a file is set apart by a `%%Page:` comment. The comment `%%Page: 2 2` may precede the description of the second page of the file. Document structure and general convention comments are covered in greater detail later in this article.

Requirements conventions—specify resources supplied or needed by a page description, as well as printer-specific features and requirements of a job. For example, the comment `%%DocumentNeededResources: font Palatino-Roman` appearing in the header section of a document indicates that the document requires the Palatino®-Roman font to print properly, but the document does not contain the font definition.

Color Separation conventions—describe document and page color usage. For example, the `%%DocumentCustomColors:` comment indicates which custom colors (also called spot colors) are used in a document, and the `%%CMYKCustomColor:` comment indicates how to represent a custom color using process color.

DSC Structure and Code Requirements

In addition to having all relevant DSC comments, a conforming page description must also adhere to the document structuring rules described primarily in section four of the specification. The following guidelines outline the basic requirements.

1. ***Distinct prolog and script.*** Organize the page description into two clearly-defined sections, prolog and script, and separate the sections with the `%%EndProlog` comment. The prolog section contains header comments, an optional defaults section, and procedure definitions. The script section contains the document set-up, page descriptions, and trailer section, which must all be self-contained.

2. ***Page independence.*** Do not generate page descriptions that rely on graphic state set-up or procedure definitions occurring on previous pages. Do not do any device setup inside page descriptions. Wrap each page description in a **save-restore** pair to help ensure page independence, and to reclaim non-global VM used during page marking.

3. ***Line length.*** Do not generate output with lines of code that exceed 255 characters. Do terminate each line with a CR (carriage return), LF (line feed), or a CR-LF pair. The line-length limitation applies to the PostScript language program as well as the DSC comments. The intent of this limitation is to allow lines of a file to be read into a 255-character buffer without overflow (Pascal strings are a common example of this sort of buffer). The exception to this rule is in-line data that is bracketed by `%%Begin(End)Data` comments. The `%%Begin(End)Data` comments allow binary image data and other binary data to be skipped over by DSC consumers.

%!PS-Adobe-3.0

4. Using **showpage***.* Call the **showpage** operator outside the page-level **save**-**restore** pair. The motivation for this is to allow applications to redefine the **showpage** operator so that it produces side effects in printer VM; for example, an application may redefine **showpage** to track the page count in a job in order to do n-up printing.

5. Operator restrictions. Avoid restricted operators in your page descriptions, or use them carefully. For example, **initgraphics**, **setmatrix**, **exitserver** are just some of the operators that you should use with discretion or not at all. See section 4.3 of the DSC specification for the complete list.

6. Loading operator definitions. Use the **load** operator to retrieve operator definitions from the dictionary stack, rather than explicitly retrieve the definition from systemdict. For example, use the code, "`/f /fill load def`" rather than "`/f systemdict /fill get def.`"

7. Modifying graphic state. Modify the graphic state; do not replace it. For example, use the **concat** operator, rather than the **setmatrix** operator to modify the current transformation matrix (CTM). Other applications which process your PostScript language output may prepend code to modify the size or placement of your document; if your code explicitly sets the graphic state, it will over-ride some of the code prepended by these document post-processors.

Why should your application generate code that conforms to the DSC?

Currently, no PostScript printers parse for DSC comments, so you may wonder why DSC structure and comments are necessary. Well, if you are printing directly to a PostScript printer, there may be no need to adhere to the document structuring conventions. However, a PostScript language file will often encounter spoolers, document post-processors, or other document managers before it reaches a PostScript output device. A *document manager* is a term used to categorize applications that manipulate PostScript language documents based on the document's structuring convention comments. A document manager may accept one or more PostScript language programs as input, and may transform the file(s) to produce altered PostScript language code as output.

A PostScript language file must conform to the DSC to receive services from a document manager. If your application creates its own PostScript language output, conforming to the DSC will ensure that your users receive all the services that a document manager has to offer. Document manager services may include page reversal, n-up printing, page range printing, spooling, error handling, and resource management, to name just a few. For a complete list of potential services, see section two of the DSC specification.

Figure 1 shows the flow of a PostScript language file from an application or driver to a printer, via a document manager.

continued on page 4

%!PS-Adobe-3.0

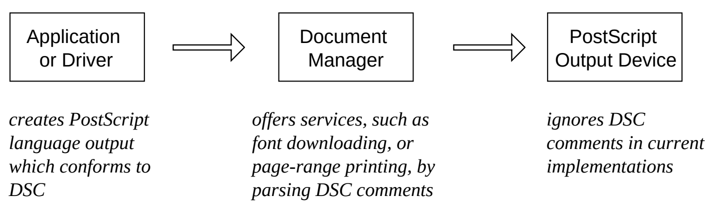

Application or Driver
Document Manager
PostScript Output Device
creates PostScript language output which conforms to DSC
offers services, such as font downloading, or page-range printing, by parsing DSC comments
ignores DSC comments in current implementations

Figure 1. Flow of PostScript language file from application or driver to printer, via a document manager.

Examples of Resource Management

If you have a PostScript language file that needs the font Minion®-Regular in order to print properly, you can put a `%%DocumentNeededResources: font Minion-Regular` comment in your file's header section, and a `%%IncludeResource: font Minion-Regular` comment later in the document to indicate where the font needs to be inserted. Then, a document manager could provide this font resource during the printing process.

Alternatively, you may include the font definition in your output between `%%BeginResource:/%%EndResource` comments, and put a corresponding `%%DocumentSuppliedResources: font Minion-Regular` comment in the header. If a spooler knows that Minion-Regular has already been loaded into the target printer's memory, then it may remove that font definition from the page description and therefore save the time it would take to transmit and define that font.

Example of Page Range Printing

If a PostScript language file follows the DSC, it is an easy task for a document manager to print a limited range of pages from that output file. For example, to print page 50 of a 100-page document, a document manager would send the prolog, document set-up, page description, and trailer sections of the document. These sections would be clearly delimited by DSC comments. Without DSC comments, proper document structure, and page independence, this service can not be made available. For example, if an application violated the DSC by making device setup calls on page 1 (after `%%Page 1 1` comment), then an attempt to print page 50 without printing page 1 may have undesirable results.

Note: If your application manipulates DSC-compliant files, you should also produce DSC-compliant output. No document manager should assume that it is the last program in the workflow of a print job or that it is talking directly to the printer.

Who parses for DSC comments?

Document managers were introduced briefly in the previous section. Document managers include print spoolers, font and resource servers, document post-processors, as well as complete systems which integrate any number of these components. Document management software ships with many high-end output devices offered by companies including Kodak, Xerox, Linotype, Agfa, Indigo, Radius, Cactus and EFI.

Attention: *Developers!*

Each month, we reserve space in our newsletter for input from our developers. Our Developers Column is a place for members of the ADA to share information with other developers.

If you would like to contribute to our Developers Column, please contact us with your ideas. We look forward to hearing from you.

%!PS-Adobe-3.0

In addition, Adobe Systems has a number of prepress applications which manipulate DSC-conforming PostScript language files. These include: Adobe OPEN,™ PressWise,® TrapWise,® Preprint® Pro, Color Central® and Print Central.™ Below we give just a few examples of the services available from some of these programs, given a DSC-compliant file. This is not a list of minimum requirements for the applications listed; it is a very limited feature list, offered to exemplify the types of services available from shipping products today.

Examples of Adobe Consumers of DSC-conforming files

Application:

• Adobe OPEN—an electronic production manager that controls the workflow of an entire PostScript prepress process. Adobe OPEN works with other prepress products to automate repetitive tasks such as trapping, imposition and image replacement.

Requirement	*Service*
`%%Title:` and `%%For:` comments	• displays title and sender of job in user-interface.

Applications:

• Adobe PrePrint Pro—a separation tool for high volume production. Adobe PrePrint Pro produces separations from composite PostScript language files.

• Adobe TrapWise—a trapping tool that automatically creates chokes and spreads for all elements in a PostScript language document.

Note: Adobe TrapWise and PrePrint Pro support EPS files and a subset of DSC-conforming PostScript language files. For more details, see the user guides for these applications.

Requirement	*Service*
`%%DocumentCustomColors:`	• displays separation choices to the user
`%%DocumentProcessColors:`	• displays separation choices to the user
`%%CMYKCustomColor:`	• spot color to process color conversion*

**Putting the correct Color Separation convention comments in your application's PostScript language output is only one step in a larger process; your output must also include special color convention operators and conform to specific guidelines to ensure its successful separation. For more information on creating PostScript language files which can be separated by page-layout or prepress applications, see technical note #5044, "Color Separation Conventions," available from the Adobe Developers Association.*

continued on page 6

%!PS-Adobe-3.0

Application:
• Adobe PressWise—Page imposition software that produces plate-ready signatures for web and sheet-fed presses. The application gives users control over page sequencing, signature design and press marks.

Requirement	*Service*
`%%Begin/EndFeature` comments	• removes device-specific code, and replaces it with appropriate code.
`%%BoundingBox:`	• positions, trims, bleeds and scales pages in an imposed form.
page independence	• re-arranges pages into signatures (also called imposition).

Applications:
• Adobe Print Central—a print spooler with features such as file and queue management and job accounting.

• Adobe Color Central—an OPI™ image server with all the features of Adobe Print Central.

Requirement	*Service*
`%%DocumentNeededFonts:` or `%%DocumentNeededResources:` and `%%IncludeResource:` comments	• Downloads needed font or resource if available.

In addition to the prepress products from Adobe, a number of third party prepress products rely on DSC in PostScript language files. DSC plays a role in network printing as well; for example, when using ATPS (AppleTalk® Print Services), Novell's Netware® software will parse a file's DSC Query comments to respond to printer feature queries made by an application or PostScript printer driver.

How may your application generate conforming output?
Figure 2 demonstrates the general comments and structure in a DSC-conforming document. If a particular section is not used (for example, a page set-up section), then the corresponding comments (`%%Begin/EndPageSetup`) can be omitted; however, if a section is present, the comments delimiting that section are required.

%!PS-Adobe-3.0

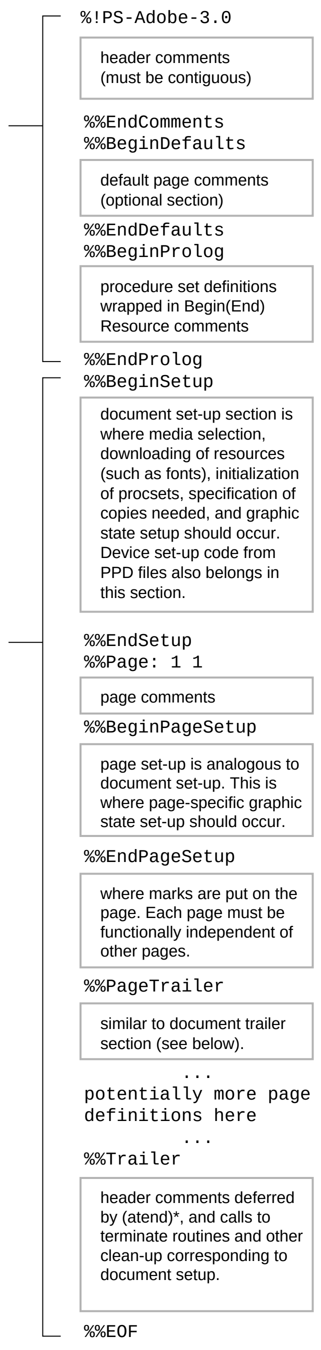
Figure 2. Structure of DSC-Conforming Document

Prolog

Executing the prolog should only result in the definition of procedure sets. It should produce no output and put no marks on the page. It should not alter the graphic state.

Script

The script can contain definitions like the prolog, but it can also modify the graphics environment, draw marks on the page, and issue showpage calls. All sections and pages in the script should be self-contained.



```
%!PS-Adobe-3.0
```

header comments (must be contiguous)

```
%%EndComments
%%BeginDefaults
```

default page comments (optional section)

```
%%EndDefaults
%%BeginProlog
```

procedure set definitions wrapped in Begin(End) Resource comments

```
%%EndProlog
%%BeginSetup
```

document set-up section is where media selection, downloading of resources (such as fonts), initialization of procsets, specification of copies needed, and graphic state setup should occur. Device set-up code from PPD files also belongs in this section.

```
%%EndSetup
%%Page: 1 1
```

page comments

```
%%BeginPageSetup
```

page set-up is analogous to document set-up. This is where page-specific graphic state set-up should occur.

```
%%EndPageSetup
```

where marks are put on the page. Each page must be functionally independent of other pages.

```
%%PageTrailer
```

similar to document trailer section (see below).

```
        ...
potentially more page
definitions here
        ...
%%Trailer
```

header comments deferred by (atend)*, and calls to terminate routines and other clean-up corresponding to document setup.

```
%%EOF
```

**atend is discussed in the PostScript Language Technology column in this issue.*

continued on page 8

%!PS-Adobe-3.0

Sample Conforming PostScript Language File

Code Sample 1 is a sample PostScript language file, which demonstrates the types of comments and code you may see in a conforming document. You may choose to use this example as an outline for your own application's output.

Code Sample 1

```
%!PS-Adobe-3.0
%%Creator: (MyApplication 1.0)
%%For: (Suzy Smith)
%%Title: (sample file)
%%CreationDate: (2 PM 5/30/95)
%%BoundingBox: 98 90 200 200
%%DocumentNeededResources: font Minion-Regular
%%DocumentSuppliedResources: procset graphics_ops 1.0 0
%%+ procset paint_ops 1.0 0
%%+ procset level_op 1.0 0
%%DocumentData: Clean7Bit
%%PageOrder: Ascend
%%DocumentProcessColors: Cyan Yellow Black
%%Pages: 2
%%Requirements: numcopies(2)
%%EndComments

%%BeginDefaults
%%PageOrientation: Portrait
%%EndDefaults

%%BeginProlog
%%BeginResource: procset graphics_ops 1.0 0
%%Title: (Graphics Operators)
%%Version: 1.0
userdict /graphics_ops 3 dict dup begin put
/m /moveto load def
/l /lineto load def
/box { % w h x y box —
    moveto
    1 index 0 rlineto
    0 exch rlineto
    neg 0 rlineto
    closepath
} def
currentdict readonly pop end
%%EndResource

%%BeginResource: procset paint_ops 1.0 0
%%Title: (Painting Operators)
%%Version: 1.0
userdict /paint_ops 2 dict dup begin pu
/f /fill load def
/s /stroke load def
currentdict readonly pop end
%%EndResource
```

%!PS-Adobe-3.0

```
%%BeginResource: procset level_op 1.0 0
%%Title: (Language Level)
%%Version: 1.0
userdict /level_op 1 dict dup begin put
/level2
   /languagelevel where
   {
      pop languagelevel 2 ge
   }{
      false
   }ifelse
def
currentdict readonly pop end
%%EndResource
%%EndProlog

%%BeginSetup
level_op begin
graphics_ops begin
paint_ops begin
userdict begin
level2 {       % setting copies under level 1
   /#copies 2 def
}{             % setting copies under level 2
   1 dict begin /NumCopies 2 def currentdict end setpagedevice
} ifelse

%%BeginFeature: *PageRegion LetterSmall
level2 {
        2 dict dup /PageSize [612 792] put dup
        /ImagingBBox [30 31 582 761] put setpagedevice
}{
       /lettersmall where {pop lettersmall} {letterR} ifelse
} ifelse
%%EndFeature
%%EndSetup

%%Page: 1 1
%%PageBoundingBox: 98 90 200 112
%%PageResources: font Minion-Regular
%%PageProcessColors: Black
%%BeginPageSetup
/pgsave save def
0 0 0 1 setcmykcolor
%%IncludeResource: font Minion-Regular
%%EndPageSetup
/Minion-Regular findfont 12 scalefont setfont
100 100 m
(Hello world) show
100 95 m 190 95 l s
pgsave restore
showpage
```

continued on page 10

%!PS-Adobe-3.0

```
%%Page: 2 2
%%PageBoundingBox: 100 100 200 200
%%PageProcessColors: Cyan Yellow
%%BeginPageSetup
/pgsave save def
0.5 0 0.5 0 setcmykcolor % sets default color for page
%%EndPageSetup
100 100 100 100 box f
pgsave restore
showpage

%%Trailer
end % userdict
end % paint_ops
end % graphics_ops
end % level_op
%%EOF
```

Comments on Code Sample 1

Some features of Code Sample 1 to note are as follows:

1. ***Placement of device-dependent code.*** The device-dependent code, taken from a PostScript Printer Description (PPD) file, is invoked in the document set-up section and is wrapped in `%%Begin(End)Feature` comments. This enables a document manager to remove this device-dependent code or replace it with code for an alternate device.

2. ***Self-contained resource definitions.*** There are three procedure sets (procsets) defined in the prolog section of the code. Note that the definition of the procset resource does not leave a dictionary on the operand stack. The appropriate place to put the procset dictionary on the stack is in the document set-up portion of the code, as shown. If your application adheres to this convention, a resource manager could take the portion of the code between `%%BeginResource:` and `%%EndResource` and download that procset to the printer or include it in another job.

3. ***General convention comments as needed.*** There are no `%%PageTrailer` comments in the file because there is no page trailer section. The page set-up portion of the code requires no corresponding clean-up section. If a page trailer section were needed, a `%%PageTrailer` comment would be required to designate the location of that portion of the page description.

4. ***Page-level color separation convention comments.*** By parsing the page-level color separation convention comments, a document manager can determine color usage on a page by page basis. An intelligent document manager could save a user a tremendous amount of money by

%!PS-Adobe-3.0

only printing film separations for the colors used on a given page. Alternatively, a spooler with access to both a color and a black and white printer could send some pages to each printer, depending on the page's color needs.

Code Sample 1 uses the `%%DocumentProcessColors:` comment because its output consists of one or more of the Cyan, Magenta, and Yellow and Black process colors. If your output contains RGB or other non-CMYK marking, this comment could be misleading to document managers and should be omitted. If your PostScript language output uses only the DeviceGray color space, you may use this comment as follows: "`%%DocumentProcessColors: Black`". As noted earlier, if you intend to create a PostScript language file that can be separated by another application, there are a number of additional contraints and conventions that must be adhered to, beyond the commenting conventions. Refer to technote #5044 for more information.

Supporting Included Files

Code Sample 1 does not address the inclusion of other DSC-conforming documents within your output file. There are three basic guidelines, with respect to the DSC, for outputing a document that contains another document:

1. Parse the imported document's DSC header comments to obtain information about the document's needs. For example, your application should determine which fonts the imported file needs, which colors it uses, and the language level (or extensions) needed to interpret the file.

2. Update your DSC header comments to reflect the needs of the included document. For example, if the included document needs the Palatino-Regular font to print properly, your output file now needs Palatino-Regular to print properly, and your header comments should contain the appropriate `%%DocumentNeededResources: font Palatino-Regular` comment to indicate this.

3. Enclose the file within `%%BeginDocument:` and `%%EndDocument` comments. These comments are necessary to allow multiple occurrences of the `%!PS-Adobe-3.0`, `%%EndProlog`, `%%Trailer`, and `%%EOF` comments in the body of a document.

For more details about what is required of applications that import EPS files or other DSC-conforming documents, please see technical note #5002, "Encapsulated PostScript File Format Specification, Version 3.0" or Appendix H of the *PostScript Language Reference Manual, Second Edition.*

continued on page 12

%!PS-Adobe-3.0

Where to go from here?

If your application outputs PostScript language code, the code should conform to the current DSC specification. DSC-conforming output will give your users flexibility, and help them save time and money, by allowing them to receive services from document managers.

As described in this article, to be a DSC-compliant document, a PostScript language file must conform to the document structuring guidelines described in the DSC specification and include all relevant DSC comments. Please see the DSC specification, version 3.0, for more details.

The Adobe PostScript SDK has a demo version of a third-party utility called DSCVerifier, which may help you discover DSC violations in your PostScript language output.The utility is available for both Macintosh® and Windows® platforms. If you have the SDK, you may find the product in *CONTENTS/TOOLS/3RDPARTY/DSCVERFY*. §

PostScript Language Technologies

In this month's feature article, entitled "%!PS-Adobe-3.0," we give examples of comments required in DSC-conforming PostScript language files. Some of the information needed for these DSC comments, such as a document's font and color usage, may not be available when the PostScript printer driver is creating its PostScript language output. Some drivers create their output code on-the-fly, which means they know little about the page being processed prior to its output. For this reason, the DSC allows some comments to be deferred to the `%%Trailer` or `%%PageTrailer` sections of a document by using the (atend) convention. To defer a comment until later in the document, list the comment in the header section with an (atend) for its argument list. The same comment and its appropriate arguments must appear in the `%%Trailer` or `%%PageTrailer` sections of the document.

While the use of (atend) is compliant with the DSC specification, its use could be problematic for some DSC consumers. For example, use of (atend) could force a spooler to process an entire file before transmitting its first page to the printer, which would require more disk space. Furthermore, some document managers may not support (atend).The (atend) argument should therefore only be used when there is no alternative.

Code Sample 1 exemplifies the use of (atend) for the header comments, `%%DocumentProcessColors:` and `%%DocumentNeededResources:`, and the page comment, `%%PageProcessColors:`.

Code Sample 1

```
%!PS-Adobe-3.0
%%Creator: (MyOtherApplication 1.0)
%%For: (Trevor Smith)
%%Title: (another sample file)
%%CreationDate: (3 PM 6/30/95)
%%BoundingBox: 100 94 190 110
%%DocumentNeededResources: (atend)
%%DocumentSuppliedResources: procset paint_ops 1.0 0
%%DocumentData: Clean7Bit
%%DocumentProcessColors: (atend)
%%Pages: 1
%%EndComments
%%EndProlog

%%Page: 1 1
%%PageOrientation: Portrait
%%PageProcessColors: (atend)
%%BeginPageSetup
/pgsave save def
%%IncludeResource: font Minion-Regular
%%EndPageSetup
/Minion-Regular findfont 12 scalefont setfont
100 100 moveto
(Hello world) show
100 95 moveto 190 95 lineto stroke
pgsave restore
showpage

%%PageTrailer
%%PageProcessColors: Black
%%Trailer
%%DocumentProcessColors: Black
%%DocumentNeededResources: font Minion-Regular
%%EOF
```

For more information about which comments support the (atend) convention and how to use it, see section 4.6 of the DSC specification.

ACROBAT column™

This month, we'll describe how plug-ins can add support for new annotation types to the Acrobat Exchange™ viewer. Annotations in the Adobe Portable Document Format (PDF) are data that are associated with a page, but are not part of the page contents. Version 2.0 of the Acrobat viewers provide two built-in annotation types—Link (used for hypertext links) and Text (used for Notes). Plug-ins can add new annotation types, which might support sound, allow document markup, or any number of other things.

Plug-ins that support new annotation types do so by providing an annotation handler — a collection of functions and data that specify and implement the annotation type's behavior. An annotation handler contains functions that perform the following:

- Draw the annotation.
- Highlight (unhighlight) the annotation when the user selects (deselects) it.
- Handle mouse clicks in the annotation.
- Handle key clicks in the annotation.
- Control the cursor shape when the cursor is within the annotation.
- Return a boolean indicating whether or not a specified point is within the annotation. This is actually implemented as two functions—one that is useful for solid rectangular annotations (such as the Note annotation), and another that is useful for annotations of other shapes (such as the Link annotation, which is a hollow rectangular border). These functions are used by the Acrobat viewer to determine which annotation the cursor is over, or in which annotation a user has clicked.
- Return the annotation's type (e.g., "Link" for Link annotations and "Text" for Note annotations). This function allows the Acrobat viewer to determine which annotation type the handler services.
- Display whatever user interface is desired to allow users to modify an annotation's properties (color, font usage, etc.).
- Return the layer in which the annotation appears. Layers are used in the Acrobat viewer to determine the order in which objects are drawn. Layers with higher numbers are drawn after layers with lower numbers. An annotation's layer may vary; for example, a Text annotation is in a higher layer when open than when closed, ensuring that closed Text annotations are never drawn on top of open ones.

Note that there are no functions to create, delete, or save annotations. The Acrobat viewer can save and delete annotations without assistance from an annotation handler because all annotations have a standardized location and general structure in PDF. To create annotations, plug-ins must add some user-interface code (commonly a toolbar button or menu item) outside of the annotation handler. This code must call an Acrobat viewer plug-in API method to create a generic empty annotation, and then populate the annotation with the appropriate data.

Because annotation data is stored in PDF files, the names used for the data and the name of the annotation type must follow the rules listed in Appendix F of Technical Note #5156, Updates to the Portable Document Format Reference Manual. These rules, which essentially state that all names must include a developer-specific prefix registered with Adobe's Developer Support group, ensure that two developers do not use the same name for different data.

In addition to the functions listed above, annotation handlers contain a set of flags that specify certain aspects of their annotation's behavior. Only one flag is currently defined. This flag specifies whether or not the Acrobat viewer should highlight text within the annotation when the user selects text in the underlying PDF document and the selection extends into the annotation. This flag is necessary because some annotation types such as the Link annotation are transparent and should permit highlighting, while others such as the Text annotation are opaque and should not.

Plug-ins register an annotation handler with the Acrobat viewer by passing a set of pointers to the annotation handler's functions and data to the Acrobat viewer. From then on, the Acrobat viewer calls the annotation handler's functions automatically to service the type of annotations the handler implements.

If the Acrobat viewer encounters an annotation for which a handler cannot be found, it either displays a generic icon for the annotation or ignores the annotation, depending on the value of the F key in the dictionary representing the annotation in the PDF file.

Custom annotations provide a way to extend the richness of information in PDF files. The Acrobat Plug-ins SDK, available through the Adobe Developers Association, contains a sample annotation handler and further information about writing custom annotation handlers.

Colophon

This newsletter was produced entirely with Adobe PostScript software on Macintosh and IBM® PC compatible computers. Typefaces used are from the Minion and Myriad® families from the Adobe Type Library.

Managing Editor:
Jennifer Cohan

Technical Editor:
Nicole Frees

Art Director:
Gail Blumberg

Designer:
Lorsen Koo

Contributors:
Tim Bienz, Nicole Frees

Part Number ADA0060 10/95

Adobe PostScript