PostScript Language Document Structuring Conventions Specification

Adobe Developer Support

Version 3.0

25 September 1992

Adobe Systems Incorporated

Adobe Developer Technologies
345 Park Avenue
San Jose, CA 95110
http://partners.adobe.com/

PN LPS5001

Contents

List of Figures

PostScript Language Document Structuring Conventions Specification

As discussed in Chapter 3 of the *PostScript Language Reference Manual, Second Edition*, the PostScript™ language standard does not specify the overall structure of a PostScript language program. Any sequence of tokens conforming to the syntax and semantics of the PostScript language is a valid program that may be presented to a PostScript interpreter for execution.

For a PostScript language program that is a page description (in other words, a description of a printable document), it is often advantageous to impose an overall program structure.

A page description can be organized as a prolog and a script, as discussed in section 2.4.2, “Program Structure” of the *PostScript Language Reference Manual, Second Edition*. The prolog contains application-dependent definitions. The script describes the particular desired results in terms of those definitions. The prolog is written by a programmer, stored in a place accessible to an application program, and incorporated as a standard preface to each page description created by the application. The script is usually generated automatically by an application program.

Beyond this simple convention, this appendix defines a standard set of document structuring conventions (DSC). Use of the document structuring conventions not only helps assure that a document is device independent, it allows PostScript language programs to communicate their document structure and printing requirements to *document managers* in a way that does not affect the PostScript language page description.

A document manager can be thought of as an application that manipulates the PostScript language document based on the document structuring conventions found in it. In essence, a document manager accepts one or more PostScript language programs as input, transforms them in some way, and produces a PostScript language program as output. Examples of document managers include print spoolers, font and other resource servers, post-processors, utility programs, and toolkits.

If a PostScript language document properly communicates its structure and requirements to a document manager, it can receive certain *printing services*. A document manager can offer different types of services to a document. If the document in question does not conform to the DSC, some or all of these services may be denied to it.

Specially formatted PostScript language comments communicate the document structure to the document manager. Within any PostScript language document, any occurrence of the character % *not* inside a PostScript language string introduces a *comment*. The comment consists of all characters between the % and the next newline, including regular, special, space, and tab characters. The scanner ignores comments, treating each one as if it were a single white-space character. DSC comments, which are legal PostScript language comments, do not affect the destination interpreter in any manner.

DSC comments are specified by two percent characters (%%) as the first characters on a line (no leading white space). These characters are immediately followed by a unique keyword describing that particular comment—again, no white space. The keyword always starts with a capital letter and is almost always mixed-case. For example:

```
%%BoundingBox: 0 0 612 792
%%Pages: 45
%%BeginSetup
```

Note that some keywords end with a colon (considered to be part of the keyword), which signifies that the keyword is further qualified by options or arguments. There should be one space character between the ending colon of a keyword and its subsequent arguments.

The PostScript language was designed to be inherently device independent. However, there are specific physical features that an output device may have that certain PostScript operators activate (in Level 1 implementations many of these operators are found in **statusdict**). Examples of device-dependent operators are **legal**, **letter**, and **setsoftwareiomode**. Use of these operators can render a document *device dependent*; that is, the document images properly on one type of device and not on others.

Use of DSC comments such as %%BeginFeature:, %%EndFeature (note that the colon is part of the first comment and that this comment pair is often referred to as %%Begin(End)Feature) and %%IncludeFeature: can help reduce device dependency if a document manager is available to recognize these comments and act upon them.

The DSC are designed to work with *PostScript printer description* (PPD) files, which provide the PostScript language extensions for specific printer features in a regular parsable format. PPD files include information about printer-specific features, and include information about the fonts built into the ROM of each printer. The DSC work in tandem with PPD files to provide a way to specify and invoke these printer features in a *device-independent* manner. For more information about PPD files, see the *PostScript Printer Description Files Specification* available from the Adobe Systems Developers' Association.

Note *Even though the DSC comments are a layer of communication beyond the PostScript language and do not affect the final output, their use is considered to be good PostScript language programming style.*

1 Using the Document Structuring Conventions

Ideally, a document composition system should be able to compose a document regardless of available resources—for example, font availability and paper sizes. It should be able to rely on the document management system at printing time to determine the availability of the resources and give the user reasonable alternatives if those resources are not available.

Realistically, an operating environment may or may not provide a document management system. Consequently, the DSC contain some redundancy. There are two philosophically distinct ways a resource or printer-specific feature might be specified:

- The document composition system *trusts* its environment to handle the resource and feature requirements appropriately, and merely specifies what its particular requirements are.

- The document composer may not know what the network environment holds or even that one exists, and *includes* the necessary resources and printer-specific PostScript language instructions within the document. In creating such a document, the document composer delimits these included resources or instructions in such a way that a document manager can recognize and manipulate them.

It is up to the software developer to determine which of these methods is appropriate for a given environment. In some cases, both may be used.

These two methods are mirrored in the DSC comments:

- Many DSC comments provide %%Begin and %%End constructs for identifying resources and printer-specific elements of a document. The document then prints regardless of whether a document manager is present or not.

- Many of the requirement conventions provide a mechanism to specify a need for some resource or printer-specific feature through the use of %%Include comments, and leave the inclusion of the resource or invocation of the feature to the document manager. This is an example of complete network cooperation, where a document can forestall some printing decisions and pass them to the next layer of document management. In general, this latter approach is the preferred one.

2 Document Manager Services

A document manager can provide a wide variety of services. The types of services are grouped into five management categories: spool, resource, error, print, and page management. The DSC help facilitate these services. A document that conforms to this specification can expect to receive any of these services, if available; one that does not conform may not receive any service. Listed below are some of the services that belong to each of these categories.

2.1 Spool Management

Spooling management services are the most basic services that a document manager can perform. A category of DSC comments known as general conventions—specifically the header comments—provide information concerning the document's creator, title, pages, and routing information.

Spooling

The basic function of spool management is to deliver the document to the specified printer or display. The document manager should establish queues for each device to handle print job traffic in an effective manner, giving many users access to one device. In addition, the document manager should notify the user of device status (busy/idle, jammed, out of paper, waiting) and queue status (held, waiting, printing). More advanced document managers can offer job priorities and delayed-time printing.

Banner and Trailer Pages

As a part of spool management, a document manager can add a banner or trailer page to the beginning or end, respectively, of each print job to separate the output in the printer bin. The document manager can parse information from the DSC comments to produce a proper banner that includes the title, creator, creation date, the number of pages, and routing information of the document.

Print Logging

If a document manager tracks the number of pages, the type of media used, and the job requirements for each document, the document manager can produce a comprehensive report on a regular basis detailing paper and printer usage. This can help a systems administrator plan paper purchases and estimate printing costs. Individual reports for users can serve as a way to bill internally for printing.

2.2 Resource Management

Resource management services deal with the inclusion, caching, and manipulation of resources, such as fonts, forms, files, patterns, and documents. A category of DSC comments, known as requirement conventions, enables a document manager to properly identify instances in the document when resources are either needed or supplied.

Resource Inclusion

Frequently used resources, such as company logos, copyright notices, special fonts, and standard forms, can take up vast amounts of storage space if they are duplicated in a large number of documents. The DSC support special %%Include comments so a document manager can include a resource at print time, saving disk space.

Supplied resources can be cached in a resource library for later use. For example, a document manager that identifies a frequently used logo while processing a page description subsequently stores the logo in a resource library. The document manager then prints the document normally. When future %%IncludeResource: comments are found in succeeding documents, the document manager retrieves the PostScript language program for the logo from the resource library. The program is inserted into the document at the position indicated by the DSC comment before the document is sent to the printer.

Resource Downloading

Another valuable service that a document manager can provide is automatically downloading frequently used resources to specific printers so those resources are available instantly. Transmission and print time of documents can be greatly reduced by using this service.

For example, the document manager judges that the Stone-Serif font program is a frequently used resource. It downloads the font program from the resource library to the printer. Later, the document manager receives a document that requests the Stone-Serif font program. The document manager knows this resource is already available in the printer and sends the document to the printer without modification. Note that the resource can be downloaded persistently into VM or onto a hard disk if the printer has one. For Level 2 interpreters, resources are found automatically by the **findresource** operator.

Resource Optimization

An intelligent document manager can alter the position of included resources within a document to optimize memory and/or resource usage. For example, if an encapsulated PostScript (EPS) file is included several times in a document, the document manager can move duplicate *procedure set definitions* (procsets) to the top of the document to reduce transmission time. If a document manager performs dynamic resource positioning, it must maintain the relative order of the resources to preserve any interdependencies among them.

2.3 Error Management

A document manager can provide advanced error reporting and recovery services. By downloading a special error handler to the printer, the document manager can detect failed print jobs and isolate error-producing lines of PostScript language instructions. It can send this information, a descriptive error message, and suggestions for solution back to the user.

There may be other instances where a document manager can recover from certain types of errors. Resource substitution services can be offered to the user. For example, if your document requests the Stone-Serif font program and this font program is not available on the printer or in the resource library, a document manager could select a similar font for substitution.

2.4 Print Management

Good print management ensures that the requested printer can fulfill the requirements of a particular document. This is a superset of the spool management spooling function, which is concerned with delivering the print job to the printer regardless of the consequences. By understanding the capabilities of a device and the requirements of a document, a document manager can provide a wide variety of print management services.

Printer Rerouting

A document manager can reroute documents based on printer availability. Heavily loaded printers can have their print jobs off-loaded to different printers in the network. The document manager can also inform a user if a printer is busy and suggest an idle printer for use as a backup.

If a specified printer cannot meet the requirements of a document (if for example, the document requests duplex printing and the printer does not support this feature), the document manager can suggest alternate printers.

For example, a user realizes that a document to be printed on a monochrome printer contains a color page. The user informs the document manager that the document should be rerouted to the color printer. Any printer-specific portions are detected by the document manager via the %%Begin(End)Feature: comments. The document manager consults the appropriate PostScript printer description (PPD) file, the printer-specific portion is replaced in the document, and the document is rerouted to the appropriate queue.

Feature Inclusion

This service is similar in concept to resource inclusion. Instead of using PostScript language instructions that activate certain features of a target printer, an application can use the %%IncludeFeature: comment to specify that a fragment of feature instructions should be included in the document at a specific point. A document manager can recognize such a request, consult the PPD file for the target printer, look for the specified feature, and insert the code into the document before sending it to the printer.

Parallel Printing

Parallel printing, another possible feature of a document manager, is especially useful for large documents or rush orders. Basically, the document manager splits the document based on the %%Page: comment, sending different pieces of the document to different printers simultaneously. The document is printed in parallel.

For example, a user requests that the first 100 pages of a document be printed in parallel on five separate printers. The document manager splits the document into five sections of 20 pages each, replicating the original prolog and document setup for each section. Also, a banner page is specified for each section to identify the pages being printed.

Page Breakout

Color and high-resolution printing are often expensive propositions. It does not make sense to send an entire document to a color printer if the document contains only one color illustration. When the appropriate comments are used, document managers can detect color illustrations and detailed drawings that need to be printed on high resolution printers, and split them from the original document. The document manager sends these pages separately to a high-resolution or color printer, while sending the rest of the document to lower-cost monochrome printers.

2.5 Page Management

Page management deals with organizing and reorganizing individual pages in the document. A category of comments known as *page comments* facilitate these services. See section 4.5, “Convention Categories,” for a thorough description of page-level comments.

Page Reversal

Some printers place output in the tray face-up, some face-down. This small distinction can be a nuisance to users who have to reshuffle output into the correct order. Documents that come out of the printer into a face-up tray should be printed last page first; conversely, documents that end up face-down should be printed first page first. A document manager can reorder pages within the document based on the %%Page: comment to produce either of these effects.

n-Up Printing

n-up, thumbnail, and signature printing all fall under this category. This enables the user to produce a document that has multiple *virtual* pages on fewer *physical* pages. This is especially useful when proofing documents, and requires less paper.

For example, suppose a user wants a proof of the first four pages of a document (two copies, because the user’s manager is also interested). Two-up printing is specified, where two virtual pages are mapped onto one physical sheet. The document manager adds PostScript language instructions (usually to the document setup section) that will implement this service.

Range Printing

Range printing is useful when documents need not be printed in their entirety. A document manager can isolate the desired pages from the document (using the %%Page: comment and preserving the prolog and document setup) before sending the new document to the printer. In the previous example, the user may want only the first four pages of the document. The document manager determines where the first four pages of the document reside and discards the rest.

Collated Printing

When using the **#copies** or **setpagedevice** features to specify multiple copies, on some printers the pages of the document emerge uncollated (1-1-1-2-2-2-3-3-3). Using the same mechanics as those for range printing, a document manager can print a group of pages multiple times and obtain collated output (1-2-3-1-2-3-1-2-3), saving the user the frustration of hand collating the document.

Underlays

Underlays are text and graphic elements, such as draft and confidential notices, headers, and images, that a document manager can add to a document so they appear on every page. By adding PostScript language instructions to the document setup, each page of the document renders the underlay before drawing the page itself.

3 DSC Conformance

The PostScript interpreter does not distinguish between PostScript language page descriptions that do or do not conform to the DSC. However, the structural information communicated through DSC comments is of considerable importance to document managers that operate on PostScript page descriptions as data. Because document managers cannot usually interpret the PostScript language directly, they must rely on the DSC comments to properly manipulate the document. It is necessary to distinguish between those documents that conform to the DSC and those that do not.

Note *In previous versions of the DSC, there were references to partially conforming documents. This term has caused some confusion and its use has been discontinued. A document either conforms to the conventions or it does not.*

3.1 Conforming Documents

A *conforming* document can expect to receive the maximum amount of services from any document manager. A conforming document is recognized by the header comment %!PS-Adobe-3.0 and is optionally followed by keywords indicating the type of document. Please see the description of this comment in section 5, “General Conventions,” for more details about optional keywords.

A fully conforming document is one that adheres to the following rules regarding syntax and semantics, document structure, and the compliance constraints. It is also strongly suggested that documents support certain printing services.

Syntax and Semantics

If a comment is to be used within a document, it must follow the syntactical and semantic rules laid out in this specification for that comment.

Consider the following *incorrect* example:

```
%%BoundingBox 43.22 50.45 100.60 143.49
```

This comment is incorrect on two counts. First, there is a colon missing from the %%BoundingBox: comment. Abbreviations for comments are not acceptable. Second, floating point arguments are used instead of the integer arguments this comment requires.

Document Structure

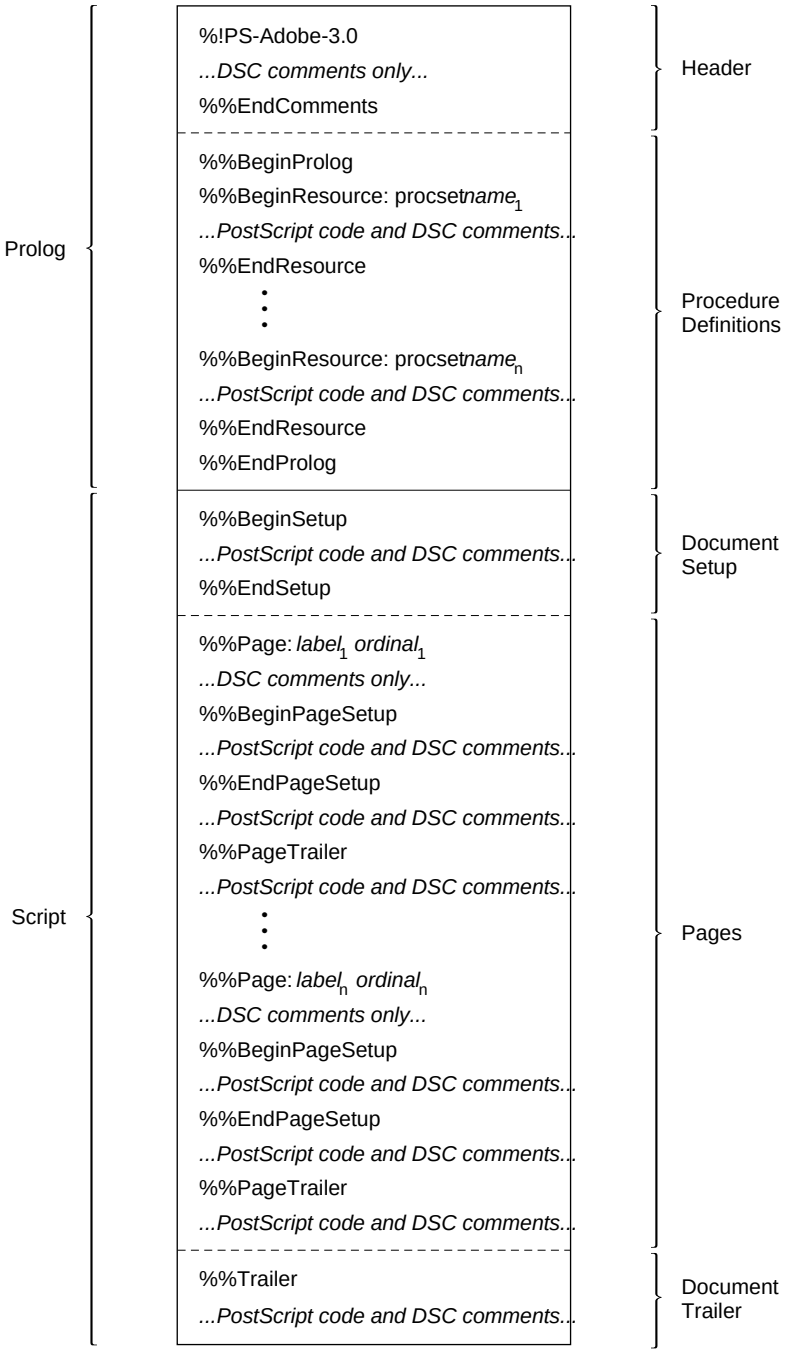
The document structure rules described in section 4, “Document Structure Rules,” must be followed. The following comments delineate the structure of the document. If there is a section of a document that corresponds to a particular comment, that comment *must* be used to identify that section of the document.

```
%!PS-Adobe-3.0
%%Pages:
%%EndComments
%%BeginProlog
%%EndProlog
%%BeginSetup
%%EndSetup
%%Page:
%%BeginPageSetup
%%EndPageSetup
%%PageTrailer
%%Trailer
%%EOF
```

For example, if there are distinct independent pages in a document, the %%Page: comment must be used at the beginning of each page to identify those pages.

Where sections of the structure are not applicable, those sections and their associated comments need not appear in the document. For example, if a document setup is not performed inside a particular document, the %%BeginSetup and %%EndSetup comments are unnecessary. Figure 1 illustrates the structure of a conforming PostScript language document.

Figure 1 *Structure of a conforming PostScript language document*



Compliance Constraints

The compliance constraints described in section 4.3, “Constraints,” including the proper use of restricted operators, *must* be adhered to.

Printing Services

There are document manager printing services (such as those described in section 2, “Document Manager Services”) that can be easily supported and add value to an application. Although it is not a requirement of a conforming document, it is *strongly suggested* that applications support these services by using the comments listed below. Note that 20 comments will ensure support of all services.

Spool Management
(Spooling, Banner and Trailer Pages, and Print Logging)

%%Creator:	%%PageMedia:
%%CreationDate:	%%PageRequirements:
%%DocumentMedia:	%%Requirements:
%%DocumentPrinterRequired:	%%Routing:
%%For:	%%Title:

Resource Management
(Resource Inclusion, Downloading, and Optimization)

%%DocumentNeededResources:	%%IncludeResource:
%%DocumentSuppliedResources:	%%Begin(End)Resource:
%%PageResources:	

Error Management
(Error Reporting and Recovery)

%%Extensions:	%%ProofMode:
%%LanguageLevel:	

Printer Management
(Printer Rerouting, Feature Inclusion, Parallel Printing, Color Breakout)

%%Begin(End)Feature:	%%IncludeFeature:
%%Begin(End)Resource:	%%IncludeResource:
%%DocumentMedia:	%%LanguageLevel:
%%DocumentNeededResources:	%%PageMedia:
%%DocumentPrinterRequired:	%%PageRequirements:
%%DocumentSuppliedResources:	%%PageResources:
%%Extensions:	%%Requirements:

Page Management
(Page Reversal, N-up Printing, Range Printing, Collation, Underlays)

%%Pages:	%%Page:
%%EndComments	%%Begin(End)PageSetup
%%Begin(End)Setup	%%PageTrailer
%%Begin(End)Prolog	%%Trailer

3.2 Non-Conforming Documents

A *non-conforming* document most likely will not receive any services from a document manager, may not be able to be included into another document, and may not be portable. In some cases, this may be appropriate; a PostScript language program may require an organization that is incompatible with the DSC. This is especially true of very sophisticated page descriptions composed directly by a programmer.

However, for page descriptions that applications generate automatically, adherence to the structuring conventions is strongly recommended, simple to achieve, and essential in achieving a transparent corporate printing network.

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.

4 Document Structure Rules

One of the most important levels of document structuring in the PostScript language is the distinction between the *document prolog* and the *document script.* The *prolog* is typically a set of procedure definitions appropriate for the set of operations a document composition system needs, and the *script* is the software-generated program that represents a particular document.

A *conforming* PostScript language document description must have a clearly defined prolog and script separated by the %%EndProlog comment.

4.1 Prolog

The prolog consists of a header section, an optional defaults subsection, and the prolog proper, sometimes known as the procedures section.

The *header section* consists of DSC comments only and describes the environment that is necessary for the document to be output properly. The end of the header section is denoted by the %%EndComments comment (see the note on header comments in section 4.5, “Convention Categories”).

The *defaults section* is an optional section that is used to save space in the document and as an aid to the document manager. The beginning of this section is denoted by the %%BeginDefaults comment. Only DSC page comments should appear in the defaults section. Information on the page-level comments that are applicable and examples of their use can be found in section 5.2, “General Body Comments” under the definition of %%Begin(End)Defaults. The end of the defaults section is indicated by the %%EndDefaults comment.

The beginning of the *procedures section* is indicated by the %%BeginProlog comment. This section is a series of procedure set (procset) definitions; each procset is enclosed between a %%BeginResource: procset and %%EndResource pair. Procsets are groups of definitions and routines appropriate for different imaging requirements.

The prolog has the following restrictions:

- Executing the prolog should define procsets only. For example, these procsets can consist of abbreviations, generic routines for drawing graphics objects, and routines for managing text and images.

- A document-producing application should almost always use the same prolog for all of its documents, or at least the prolog should be drawn from a pool of common procedure sets. The prolog should always be constructed in a way that it can be removed from the document and downloaded only once into the printer. All subsequent documents that are downloaded with this prolog stripped out should still execute correctly.

- No output can be produced while executing the prolog, no changes can be made to the graphics state, and no marks should be made on the page.

4.2 Script

The document *script* consists of three sections: a document setup section, page sections, and a document trailer.

- The *document setup* section is denoted by the %%Begin(End)Setup comments. The document setup should consist of procedure calls for invoking media selections (for example, setting page size), running initialization routines for procsets, downloading a font or other resource, or setting some aspect of the graphics state. This section should appear after the %%EndProlog comment, but before the first %%Page: comment.

- The *pages* section of the script consists of 1 to *n* pages, each of which should be *functionally independent* of the other pages. This means that each page should be able to execute in any order and may be physically rearranged, resulting in an identical document as long as the information within it is the same, but with the physical pages ordered differently. A typical example of this page reordering occurs during a page-reversal operation performed by a document manager.

  The start of each page is denoted by the %%Page: comment and can also contain a %%Begin(End)PageSetup section (analogous to the document setup section on a page level), and an optional %%PageTrailer section (similar to the document trailer). In any event, each page will contain between the setup and the trailer sections the PostScript language program necessary to mark that page.

- The *document trailer* section is indicated by the %%Trailer comment. PostScript language instructions in the trailer consists of calls to termination routines of procedures and post-processing or cleanup instructions. In addition, any header comments that were deferred using the (atend) notation will be found here. See section 4.6, “Comment Syntax Reference,” for a detailed description of (atend).

There are generally few restrictions on the script. It can have definitions like the prolog and it can also modify the graphics environment, draw marks on the page, issue **showpage**, and so on. There are some PostScript language operators that should be avoided or at least used with extreme caution. A thorough discussion of these operators can be found in Appendix I of the *PostScript Language Reference Manual, Second Edition*.

The end of a document should be signified by the %%EOF comment.

4.3 Constraints

There are several constraints on the use of PostScript language operators in a conforming document. These constraints are detailed below and are not only applicable to documents that conform to the DSC. Even a *non-conforming* document is much more portable across different PostScript interpreters if it observes these constraints.

Page Independence

Pages should not have *any* inter-dependencies. Each page may rely on certain PostScript language operations defined in the document prolog or in the document setup section, but it is not acceptable to have any graphics state set in one page of a document on which another page in the same document relies on. It is also risky to reimpose or rely on a state defined in the document setup section; the graphics state should only be added to or modified, not reimposed. See Appendix I of the *PostScript Language Reference Manual, Second Edition* for more details on proper preservation of the graphics state with operators like **settransfer**.

Page independence enables a document manager to rearrange the document’s pages physically without affecting the execution of the document description. Other benefits of page independence include the ability to print different pages in parallel on more than one printer and to print ranges of pages. Also, PostScript language previewers need page independence to enable viewing the pages of a document in arbitrary order.

For the most part, page independence can be achieved by placing a **save-restore** pair around each page, as shown below:

```
%!PS-Adobe-3.0
...Header comments, prolog definitions, document
setup...
%%Page: cover 1
%%BeginPageSetup
/pgsave save def
...PostScript language instructions to perform page
setup...
%%EndPageSetup
...PostScript language instructions to mark page 1...
pgsave restore
showpage
...Rest of the document...
%%EOF
```

The **save**-**restore** pair will also reclaim any non-global VM used during the page marking (for example, text strings).

Note *If pages must have interdependencies, the* %%PageOrder: Special *comment should be used. This ensures that a document manager will not attempt to reorder the pages.*

Line Length

To provide compatibility with a large body of existing application and document manager software, a conforming PostScript language document description *does not* have lines exceeding 255 characters, excluding line-termination characters. The intent is to be able to read lines into a 255-character buffer without overflow (Pascal strings are a common example of this sort of buffer).

The PostScript interpreter imposes no constraints as to where line breaks occur, even in string bodies and hexadecimal bitmap representations. This level of conformance should not pose a problem for software development. Any document structuring comment that needs to be continued on another line to avoid violating this guideline should use the %%+ notation to indicate that a comment line is being continued (see %%+ in section 5.2, “General Body Comments”).

Line Endings

Lines *must* be terminated with one of the following combinations of characters: CR, LF, or CR LF. CR is the carriage-return character and LF is the line-feed character (decimal ASCII 13 and 10, respectively).

Use of showpage

To reduce the amount of VM used at any point, it is common practice to delimit PostScript language instructions used for a particular page with a **save**-**restore** pair. See the page-independence constraint for an example of **save**-**restore** use.

If the **showpage** operator is used in combination with **save** and **restore**, the **showpage** should occur *after* the page-level **restore** operation. The motivation for this is to redefine the **showpage** operator so it has side effects in the printer VM, such as maintaining page counts for printing *n*-up copies on one sheet of paper. If **showpage** is executed within the confines of a page-level **save**-**restore**, attempts to redefine **showpage** to perform extra operations will not work as intended. This also applies to the **BeginPage** and **EndPage** parameters of the **setpagedevice** dictionary. The above discussion also applies to **gsave**-**grestore** pairs.

Document Copies

In a conforming document, the number of copies *must* be modified in the document setup section of the document (see %%BeginSetup and %%EndSetup). Changing the number of copies within a single page automatically breaks the page independence constraint. Also, using the **copypage** operator is not
recommended because doing so inhibits page independence. If multiple copies of a document are desired, use the **#copies** key or the **setpagedevice** operator.

In Level 1 implementations, the **#copies** key can be modified to produce multiple copies of a document as follows:

```
%!PS-Adobe-3.0
%%Pages: 23
%%Requirements: numcopies(3) collate
%%EndComments
...Prolog with procset definitions...
%%EndProlog
%%BeginSetup
/#copies 3 def
%%EndSetup
...Rest of the Document (23 virtual pages)...
%%EOF
```

In Level 2 implementations, the number of copies of a document can be set using the **setpagedevice** operator as follows:

```
<< /NumCopies 3 >> setpagedevice
```

The %%Pages: comment should not be modified if the number of copies is set, as it represents the number of unique virtual pages in the document. However, the %%Requirements: comment should have its numcopies option modified, and the collate option set, if applicable.

Restricted Operators

There are several PostScript language operators intended for system-level jobs that are not appropriate in the context of a page description program. Also, there are operators that impose conditions on the graphics state directly instead of modifying or concatenating to the existing graphics state. However, improper use of these operators may cause a document manager to process a document incorrectly. The risks of using these operators involve either rendering a document device dependent or unnecessarily inhibiting constructive post-processing of document files for different printing needs—for example, embedding one PostScript language document within another.

In addition to all operators in **statusdict** and the operators in **userdict** for establishing an imageable area, the following operators should be used carefully, or not at all, in a PostScript language page description:

banddevice	**framedevice**	**quit**	**setpagedevice**
clear	**grestoreall**	**renderbands**	**setscreen**
cleardictstack	**initclip**	**setglobal**	**setshared**
copypage	**initgraphics**	**setgstate**	**settransfer**
erasepage	**initmatrix**	**sethalftone**	**startjob**
exitserver	**nulldevice**	**setmatrix**	**undefinefont**

For more specific information on the proper use of these operators in various situations, see Appendix I of the *PostScript Language Reference Manual, Second Edition*.

There are certain operators specific to the Display PostScript system that are not part of the Level 1 and Level 2 implementations. These operators are for display systems only and *must not* be used in a document. This is a much more stringent restriction than the above list of restricted operators, which may be used with extreme care. For a complete list see section A.1.2, “Display PostScript Operators, of the *PostScript Language Reference Manual, Second Edition*.”

4.4 Parsing Rules

Here are a few explicit rules that can help a document manager parse the DSC comments:

- In the interest of forward compatibility, any comments that are not recognized by the parser should be ignored. Backward compatibility is sometimes difficult, and it may be helpful to develop an "upgrading parser" that will read in documents conforming to older versions of the DSC and write out DSC version 3.0 conforming documents.

- Many comments have a colon separating the comment keyword from its arguments. This colon is not present in all comment keywords (for example, %%EndProlog) and should be considered part of the keyword for parsing purposes. It is *not* an optional character.

- Comments with arguments (like %%Page:) should have a space separating the colon from the first argument. Due to existing software, this space must be considered optional.

- "White space" characters within comments may be either spaces or tabs (decimal ASCII 32 and 9, respectively).

- Comment keywords are case-sensitive, as are all of the arguments following a comment keyword.

- The character set for comment keywords is limited to printable ASCII characters. The keywords only contain alphabetic characters and the :, !, and ? characters. The arguments may include any character valid in the PostScript language character set, especially where procedure names, font names, and strings are represented. See the definition of the <*text*> elementary type for the use of the \ escape mechanism.

- When looking for the %%Trailer comment (or any (atend) comments), allow for nested documents. Observe %%BeginDocument: and %%EndDocument comments as well as %%BeginData: and %%EndData.

- In the case of multiple header comments, the *first* comment encountered is considered to be the truth. In the case of multiple *trailer* comments (those comments that were deferred using the (atend) convention), the last comment encountered is considered to be the truth. For example, if there are two %%Requirements: comments in the header of a document, use the first one encountered.

- Header comments can be terminated explicitly by an instance of %%EndComments, or implicitly by any line that does not begin with %*X*, where *X* is any printable character except space, tab, or newline.

- The order of some comments in the document is significant, but in a section of the document they may appear in any order. For example, in the header section, %%DocumentResources:, %%Title:, and %%Creator: may appear in any order.

- Lines must never exceed 255 characters, and line endings should follow the line ending restrictions set forth in section 4.3, “Constraints.”

- If a document manager supports resource or feature inclusion, at print time it should replace %%Include comments with the resource or feature requested. This resource or feature code should be encapsulated in %%Begin and %%End comments upon inclusion. If a document manager performs resource library extraction, any resources that are removed, including their associated %%Begin and %%End comments, should be replaced by equivalent %%Include comments.

4.5 Convention Categories

The DSC comments are roughly divided into the followingsix categories of conventions:

- General conventions

- Requirement conventions

- Color separation conventions

- Query conventions

- Open structuring conventions

- Special conventions

Typically, some subsets of the general, requirement, and color separation conventions are used consistently in a particular printing environment. The DSC have been designed with maximum flexibility in mind and with a minimum amount of interdependency between conventions. For example, one may use only general conventions in an environment where the presence of a document manager may not be guaranteed, or may use the requirement conventions on a highly spooled network.

General conventions delimit the various structural components of a PostScript language page description, including its prolog, script, and trailer, and where the page breaks fall, if there are any. The general convention comments include document and page setup information, and they provide a markup convention for noting the beginning and end of particular pieces of the page description that might need to be identified for further use.

Requirement conventions are comments that suggest document manager action. These comments can be used to specify resources the document supplies or needs. Document managers may make decisions based on resource frequency (those that are frequently used) and load resources permanently into the printer, download them before the job, or store them on a printer's hard disk, thus reducing transmission time.

Other requirement comments invoke or delimit printer-specific features and requirements, such as paper colors and weights, collating order, and stapling. The document manager can replace printer-specific PostScript language fragments based on these comments when rerouting a print job to another printer, by using information in the PostScript printer description (PPD) file for that printer.

Color separation conventions are used to complement the color extensions to the PostScript language. Comments typically identify PostScript language color separation segments in a page, note custom color ratios (RGB or CMYK), and list document and page level color use.

Query conventions delimit parts of a PostScript language program that query the current state or characteristics of a printer, including the availability of resources (for example, fonts, files, procsets), VM, and any printer-specific features and enhancements. The type of program that uses this set of conventions is usually interactive—that is, one that expects a response from the printer. This implies that document managers should be able to send query jobs to a printer, and route an answer back to the application that issued the query. Query conventions should only be used in %!PS-Adobe-3.0 Query jobs.

Open structuring conventions are user-defined conventions. Section 9, "Open Structuring Conventions," provides guidelines for creating these vendor-specific comments.

Special conventions include those comments that do not fall into the above categories.

The general, requirement, and color separation conventions can be further broken down into three classes: header comments, body comments, and page comments.

Header Comments

Header comments appear first in a document file, before any of the executable PostScript language instructions and before the procedure definitions. They may be thought of as a table of contents. In order to simplify a document manager's job in parsing these header comments, there are two rules that apply:

- If there is more than one instance of a header comment in a document file, *the first one encountered takes precedence.* This simplifies nesting documents within one another without having to remove the header comments.

- *Header comments must be contiguous*. That is, if a document manager comes across a line that does not begin with %, the document manager may quit parsing for header comments. The comments may also be ended *explicitly* with the %%EndComments convention.

All instances of lines beginning with %! *after the first instance* are ignored by document managers, although to avoid confusion, this notation should not appear twice within the block of header comments (see %%BeginDocument: and %%EndDocument for examples of embedded documents).

Body Comments

Body comments may appear anywhere in a document, except the header section. They are designed to provide structural information about the organization of the document file and should match any related information provided in the header comments section. They generally consist of %%Begin and %%End constructs to delimit specific components of the document file, such as procsets, fonts, or emulation code, and %%Include comments that request the document manager to take action when encountering the comment, such as including a document, resource, or printer-specific fragment of code.

Page Comments

Page comments are page-level structure comments. They should not span across page boundaries (see the exception below). That is, a page comment applies only to the page in which it appears. The beginning of a page should be noted by the %%Page: comment. The other page comments are similar to their corresponding header comments (for example, %%BoundingBox: vs. %%PageBoundingBox:), except for %%Begin or %%End comments that are more similar to body comments in use (e.g., %%Begin(End)Setup vs. %%Begin(End)PageSetup).

Note *Some page comments that are similar to header comments can be used in the defaults section of the file to denote default requirements or media for all pages. See the* %%Begin(End)Defaults *comments for a more detailed explanation.*

4.6 Comment Syntax Reference

Before describing the DSC comments, it is prudent to specify the syntax with which they are documented. This section introduces a syntax known as Backus-Naur form (BNF) that helps eliminate syntactical ambiguities and helps comprehend the comment definitions. A brief explanation of the BNF operators is given in Table 1. The following section discusses elementary types, which are used to specify the keywords and options of the DSC comments.

Table 1 *Explanation of BNF operators*

BNF Operator	*Explanation*
<token>	This indicates a token item. This item may comprise other tokens or it may be an elementary type (see below).
::=	Literally means “is defined as.”
[*expression*]	This indicates that the expression inside the brackets is optional.
{ *expression* }	The braces are used to group expressions or tokens into single expressions. It is often used to denote parsing order (it turns the expression inside the braces into a single token).
<token> ...	The ellipsis indicates that one or more instances of *<token>* can be specified.
|	The | character literally means “or” and delimits alternative expressions.

Elementary Types

An *elementary* or *base* type is a terminating expression. That is, it does not reference any other tokens and is considered to be a base on which other expressions are built. For the sake of clarity, these base types are defined here in simple English, without the exhaustive dissection that BNF normally requires.

(atend)
Some of the header and page comments can be deferred until the end of the file (that is, to the %%Trailer section) or to the end of a page (that is, the %%PageTrailer section). This is for the benefit of application programs that generate page descriptions on-the-fly. Such applications might not have the necessary information about fonts, page count, and so on at the beginning of generating a page description, but have them at the end. If a particular comment is to be deferred, it must be listed in the header section with an (atend) for its argument list. A comment with the same keyword and its appropriate arguments *must* appear in the %%Trailer or %%PageTrailer sections of the document.

The following comments support the (atend) convention:

%%BoundingBox:	%%DocumentSuppliedProcSets:
%%DocumentCustomColors:	%%DocumentSuppliedResources:
%%DocumentFiles:	%%Orientation:
%%DocumentFonts:	%%Pages:
%%DocumentNeededFiles:	%%PageBoundingBox:
%%DocumentNeededFonts:	%%PageCustomColors:
%%DocumentNeededProcSets:	%%PageFiles:
%%DocumentNeededResources:	%%PageFonts:
%%DocumentProcSets:	%%PageOrder:
%%DocumentProcessColors:	%%PageOrientation:
%%DocumentSuppliedFiles:	%%PageProcessColors:
%%DocumentSuppliedFonts:	%%PageResources:

Note *Page-level comments specified in the defaults section of the document cannot use the* (atend) *syntax to defer definition of their arguments.* (atend) *can only be used in the header section and within individual pages.*

In Example 1, the bounding box information is deferred until the end of the document:

Example 1

```
%!PS-Adobe-3.0
...Document header comments...
%%BoundingBox: (atend)
%%EndComments
...Rest of the document...
%%Trailer
%%BoundingBox: 0 0 157 233
...Document clean up...
%%EOF
```

<filename>

A *filename* is similar to the <*text*> elementary type in that it can comprise any printable character. It is usually very operating system specific. The following example comment lists four different files:

```
%%DocumentNeededResources: file /usr/smith/myfile.epsf
%%+ file (Corporate Logo \042large size\042) (This is (yet) another file)
%%+ file C:\LIB\LOGO.EPS
```

Note that the backslash escape mechanism is only supported inside parentheses. It can also be very convenient to list files on separate lines using the continuation comment %%+.

<fontname>

A *fontname* is a variation of the simple text string (see <*text*>). Because font names cannot include blanks, font names are considered to be delimited by blanks. In addition, the \ escape mechanism is not supported. The following example comment uses five font names:

```
%%DocumentNeededResources: font Times-Roman Palatino-Bold
%%+ font Helvetica Helvetica-Bold NewCenturySchoolbook-Italic
```

The font name does not start with a slash character (/) as it does in the PostScript language when you are specifying the font name as a literal.

<formname>

A *formname* is the PostScript language object name of the form as used by the **defineresource** operator. It is a simple text string as defined by the <*text*> elementary type.

<int>

An *integer* is a non-fractional number that may be signed or unsigned. There are practical limitations for an integer's maximum and minimum values (see Appendix B of the *PostScript Language Reference Manual, Second Edition*).

<procname> ::= <*name*> <*version*> <*revision*>
<*name*> ::= <*text*>
<*version*> ::= <*real*>
<*revision*> ::= <*uint*>

A *procname* token describes a procedure set (procset), which is a block of PostScript language definitions. A procset is labeled by a text string describing its contents and a version number. A procset *version* may undergo several revisions, which is indicated by the *revision* number. Procset names should be descriptive and meaningful. It is also suggested that the corporate name and application name be used as part of the procset name to reduce conflicts, as in this example:

```
(MyCorp MyApp - Graphic Objects) 1.1 0
Adobe-Illustrator-Prolog 2.0 1
```

The *name*, *version*, and *revision* fields should uniquely identify the procset. If a version numbering scheme is not used, these fields should still be filled with a dummy value of 0.

The *revision* field should be taken to be upwardly compatible with procsets of the same *version* number. That is, if myprocs 1.0 0 is requested, then myprocs 1.0 2 should be compatible, although the converse (backward compatibility) is not necessarily true. If the *revision* field is not present, a procset may be substituted as long as the *version* numbers are equal. Different versions of a procset may not be upwardly compatible and should not be substituted.

<patternname>

A *patternname* is the PostScript language object name of the pattern as used by the **defineresource** operator. It is a simple text string as defined by the <*text*> elementary type.

<real>

A *real* number is a fractional number that may be signed or unsigned. There are practical limitations on the maximum size of a real (see Appendix B of the *PostScript Language Reference Manual, Second Edition*). Real numbers may or may not include a decimal point, and exponentiation using either an ‘E’ or an ‘e’ is allowed. For example,

```
-.002   34.5   -3.62   123.6e10   1E-5   -1.   0.0
```

are all valid real numbers.

<resource> ::= font <*fontname*> | file <*filename*> |
procset <*procname*> | pattern <*patternname*> |
form <*formname*> | encoding <*vectorname*>
<resources> ::= font <*fontname*> ... | file <*filename*> ... |
procset <*procname*> ... | pattern <*patternname*> ... |
form <*formname*> ... | encoding <*vectorname*> ...

A *resource* is a PostScript object, referenced by name, that may or may not be available to the system at any given time. Times-Roman is the name of a commonly available resource. The name of the resource should be the same as the name of the PostScript object—in other words, the same name used when using the **defineresource** operator.

Note *Although files are not resources in the PostScript language sense, they can be thought of as a resource when document managers are dealing with them.*

<text>

A *text string* comprises any printable characters and is usually considered to be delimited by blanks. If blanks or special characters are desired inside the text string, the entire string should be enclosed in parentheses. Document managers parsing text strings should be prepared to handle multiple parentheses. Special characters can be denoted using the PostScript language string \ escape mechanism.

The following are examples of valid DSC text strings:

```
Thisisatextstring
(This is a text string with spaces)
(This is a text string (with parentheses))
(This is a special character \262 using the \\ mechanism)
```

It is a good idea to enclose numbers that should be treated as text strings in parentheses to avoid confusion. For example, use (1040) instead of 1040.

The sequence () denotes an empty string.

Note that a text string must obey the 255 character line limit as set forth in section 3, “DSC Conformance.”

<textline>

This is a modified version of the <*text*> elementary type. If the first character encountered is a left parenthesis, it is equivalent to a <*text*> string. If not, the token is considered to be the rest of the characters on the line until end of line is reached (some combination of the CR and LF characters).

<uint>

An *unsigned integer* is a non-fractional number that has no sign. There are practical limitations for an unsigned integer's maximum value, but as a default it should be able to range between 0 and twice the largest integer value given in Appendix B of the *PostScript Language Reference Manual, Second Edition*.

<vectorname>

A *vectorname* denotes the name of a particular encoding vector and is also a simple text string. It should have the same name as the encoding vector the PostScript language program uses. Examples of encoding vector names are **StandardEncoding** and **ISOLatin1Encoding**.

5 General Conventions

The general conventions are the most basic of all the comments. They impart general information, such as the bounding box, language level, extension usage, orientation, title of the document, and other basics. There are comments that are used to impart structural information (end of header, setup, page breaks, page setup, page trailer, trailer) that are the keys to abiding by the document structure rules of 3, “DSC Conformance.” Other general comments are used to identify special sections of the document, including binary and emulation data, bitmap previews, and page level objects.

5.1 General Header Comments

%!PS-Adobe-3.0 <keyword>
<*keyword*> ::= EPSF-3.0 | Query | ExitServer | Resource-<*restype*>
<*restype*> ::= Font | File | ProcSet | Pattern | Form | Encoding

This comment differs from the previous %!PS-Adobe-2.1 comment only in version number. It indicates that the PostScript language page description fully conforms to the DSC version 3.0. This comment must occur as the *first* line of the PostScript language file.

There are four *keywords* that may follow the %!PS-Adobe-3.0 comment on the same line. They flag the entire print job as a particular type of job so document managers may immediately switch to some appropriate processing mode. The following job types are recognized:

- EPSF—The file is an *Encapsulated PostScript file*, which is primarily a PostScript language file that produces an illustration. The EPS format is designed to facilitate including these illustrations in other documents. The exact format of an EPS file is described in the *PostScript Document Structuring Conventions Specifications* available from the Adobe Systems Devlopers’ Association.

- Query—The entire job consists of PostScript language queries to a printer from which replies are expected. A systems administrator or document manager is likely to create a query job. See section 8, “Query Conventions.”

- ExitServer—This flags a job that executes the **exitserver** or **startjob** operator to allow the contents of the job to persist within the printer until it is powered off. Some document managers require this command to handle these special jobs effectively. See the discussion of **exitserver** under %%Begin(End)ExitServer.

- Resource—As a generalization of the idea of Level 2 resources, files that are strictly resource definitions (fonts, procsets, files, patterns, forms) should start with this comment and keyword. For example, a procset resource should start with the %!PS-Adobe-3.0 Resource-ProcSet comment.

Fonts are resources, as well, but most fonts use one of two different header comments: %!PS-AdobeFont-1.0 and %!FontType1-1.0. In the future, fonts conforming to this specification should use the %!PS-Adobe-3.0 Resource-Font comment.

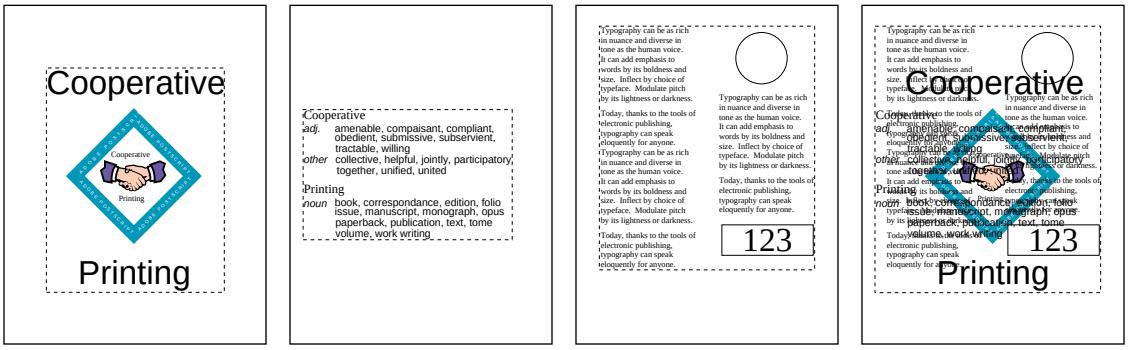
Note *Document composition programs should not use these keywords when producing a document intended for printing or display. Instead, they should use only the* %!PS-Adobe-3.0 *comment. Illustration applications may use the* EPSF-3.0 *keyword.*

%%BoundingBox: **{ <llx> <lly> <urx> <ury> } | (atend)**

<llx> ::= *<int>* (Lower left x coordinate)
<lly> ::= *<int>* (Lower left y coordinate)
<urx> ::= *<int>* (Upper right x coordinate)
<ury> ::= *<int>* (Upper right y coordinate)

This comment specifies the bounding box that encloses all marks painted on all pages of a document. That is, it must be a “high water mark” in all directions for marks made on any page. The four arguments correspond to the lower left (*llx*, *lly*) and upper right corners (*urx*, *ury*) of the bounding box in the *default user coordinate system* (PostScript units). See also the %%PageBoundingBox: comment.

Figure 2 *Determining the document bounding box*



Page 1 bounding box *Page 2 bounding box* *Page 3 bounding box* *Document bounding box*

%%Copyright: **<***textline***>**

This comment details any copyright information associated with the document or resource.

%%Creator: **<***textline***>**

This comment indicates the document creator, usually the name of the document composition software.

%%CreationDate: **<***textline***>**

This comment indicates the date and time the document was created. Neither the date nor time need be in any standard format. This comment is meant to be used purely for informational purposes, such as printing on banner pages.

%%DocumentData: **Clean7Bit | Clean8Bit | Binary**

This header comment specifies the type of data, usually located between %%Begin(End)Data: comments, that appear in the document. It applies only to data that are part of the document itself, not bytes that are added by communications software—for example, an EOF character marking the end of a job, or XON/XOFF characters for flow control. This comment warns a print manager, such as a spooler, to avoid communications channels that reserve the byte codes used in the document. A prime example of this is a serial channel, which reserves byte codes like 0x04 for end of job and 0x14 for status request.

There are three ranges of byte codes defined:

- Clean7Bit—The page description consists of only byte codes 0x1B to 0x7E (ESC to '~'), 0x0A (LF), 0x0D (CR), and 0x09 (TAB). Whenever 0x0A and/or 0x0D appear, they are used as end-of-line characters. Whenever 0x09 appears, it is used as a tab character (i.e. whitespace).

- Clean8Bit—The same as Clean7Bit, but the document may also contain byte codes 0x80–0xFF.

- Binary—Any byte codes from 0x00–0xFF may appear in the document.

The header section of the document (up to %%EndComments) must always consist of Clean7bit byte codes so it is universally readable. If the application declares the document to be Clean7Bit or Clean8Bit, it is responsible for transforming any byte codes that fall outside the acceptable range back into the acceptable range. Byte codes within character strings may be escaped—for example, a 0x05 may be written (\005).

Documents with Clean7Bit data may be transmitted to a PostScript interpreter over a serial line with 7 data bits. Documents with Clean8Bit data may be transmitted to a PostScript interpreter over a serial line with 8 data bits. Documents with Binary data cannot be transmitted over a serial line because they may use byte codes reserved by the communications protocol. However, they may be transmitted via a transparent protocol, such as LocalTalk.

%%Emulation: **<*mode*> ...**
<*mode*> ::= diablo630 | fx100 | lj2000 | hpgl | impress | hplj | ti855

This comment indicates that the document contains an invocation of the stated emulator. This allows a document manager to route the document to a printer that supports the correct type of emulation. See %%Begin(End)Emulation: for more details.

%%EndComments (no keywords)

This comment indicates an explicit end to the header comments of the document. Because header comments are contiguous, any line that does not begin with %X where X is any printable character except space, tab, or newline implicitly denotes the end of the header section.

```
%!PS-Adobe-3.0
%%Title: (Example of Header Comment Termination)
...More header comments...
%%DocumentResources: font Sonata
%GBDNodeName: smith@atlas.com
% This line implicitly denotes the end of the header
   section.
```

%%Extensions: **<*extension*> ...**
<*extension*> ::= DPS | CMYK | Composite | FileSystem

This comment indicates that in order to print properly, the document requires a PostScript Level 1 interpreter that supports the listed PostScript language extensions. The document manager can use this information to determine whether a printer can print the document or to select possible printers for rerouting the document. A list of operator sets specific to each extension is in Appendix A of the *PostScript Language Reference Manual, Second Edition***.**

- DPS—The document contains operators defined in the PostScript language extensions for the Display PostScript system. Most of these operators are available in Level 2 implementations. See Appendix A of the *PostScript Language Reference Manual, Second Edition* for a list of operators that are present only in Display PostScript implementations.

- CMYK—The document uses operators defined in the PostScript language color extensions. Note that this is different from the %%Requirements: color comment, in that it specifies that the PostScript interpreter must be able to understand the CMYK color operators. It does not specify that the printer must be capable of producing color output.

- Composite—The document uses operators defined in the PostScript language composite font extensions.

- FileSystem—This keyword should be used if the document performs file system commands. Note that certain file operators are already available under the basic implementation of the PostScript language. See Appendix A of the *PostScript Language Reference Manual, Second Edition* for a list of those operators that are specifically part of the file system extensions to Level 1 implementations.

The %%Extensions: comment must be used if there are operators in the document specific to a particular extension of the PostScript language. However, documents that provide conditional Level 1 emulation do not need to use this comment. Also, if the document uses Level 2 operators, use the %%LanguageLevel: comment instead.

%%For: <textline>

This comment indicates the person and possibly the company name for whom the document is being printed. It is frequently the "user name" of the individual who composed the document, as determined by the document composition software. This can be used for banner pages or for routing the document after printing.

%%LanguageLevel: <*uint*>

This comment indicates that the document contains PostScript language operators particular to a certain level of implementation of the PostScript language. Currently, only Level 1 and Level 2 are defined.

This comment *must* be used if there are operators in the document specific to an implementation of the PostScript language above Level 1. However, documents that provide conditional Level 1 emulation (for example, Level 1 emulation of the Level 2 operators used) need not use this comment. See Appendix D of the *PostScript Language Reference Manual, Second Edition* for emulation and compatibility strategies.

Level 2 operators are essentially a superset of the DPS, CMYK, Composite, and FileSystem language extensions. If a language level of 2 is specified, the individual extensions need not be specified. That is, use of both the %%LanguageLevel: and %%Extensions: comments is not necessary; one or the other is sufficient. See the %%Extensions: comment.

Note *To enable a document to be output to as many interpreters as possible, a document composition application should determine the minimum set of extensions needed for the document to print correctly. It is poor practice to use the* %%LanguageLevel: *comment when an* %%Extensions: *comment would have been able to encompass all of the operators used in the document.*

%%Orientation: **{ <*orientation*> ... } | (atend)**
<*orientation*> ::= Portrait | Landscape

This comment indicates the orientation of the pages in the document. It can be used by previewing applications and post-processors to determine how to orient the viewing window. A *portrait* orientation indicates that the longest edge of the paper is parallel to the vertical (y) axis. A *landscape* orientation indicates that the longest edge of the paper is parallel to the horizontal (x) axis. If more than one orientation applies to the document, an individual page should specify its orientation by using the %%PageOrientation: comment.

%%Pages: **<*numpages*> | (atend)**
<*numpages*> ::= <*uint*> (Total number of pages)

This comment defines the number of *virtual* pages that a document will image. This may be different from the number of *physical* pages the printer prints (the **#copies** key or **setpagedevice** operator and other document manager features may reduce or increase the physical number of pages). If the document produces *no* pages (for instance, if it represents an included illustration that does not use **showpage**), the page count should be 0. See also the %%Page: comment.

In previous specifications, it was valid to include an optional *page order* number after the number of pages. Its use is now discouraged because of problems with the (atend) syntax (one might know the page order before one knows the number of pages). Please use the %%PageOrder: comment to indicate page order.

%%PageOrder: **<*order*> | (atend)**
<*order*> ::= Ascend | Descend | Special

The %%PageOrder: comment is intended to help document managers determine the order of pages in the document file, which in turn enables a document manager optionally to reorder the pages. This comment can have three page orders:

- Ascend—The pages are in ascending order—for example, 1-2-3-4-5.
- Descend—The pages of the document are in descending order—for example, 5-4-3-2-1.
- Special—Indicates that the document is in a *special* order—for example, signature order.

The distinction between a page order of Special and no page order at all is that in the absence of the %%PageOrder comment, any assumption can be made about the page order, and the document manager permits any reordering of the page. However, if the page order comment is Special, the pages must be left intact in the order given.

%%Routing: **<***textline***>**

This comment provides information about how to route a document back to its owner after printing. At the discretion of the system administrator, it may contain information about mail addresses or office locations.

%%Title: **<***textline***>**

This comment provides a text title for the document that is useful for printing banner pages and for routing or recognizing documents.

%%Version: **<***version***> <***revision***>**
<*version*> ::= <*real*>
<*revision*> ::= <*uint*>

This comment can be used to note the version and revision number of a document or resource. A document manager may wish to provide version control services, or allow substitution of compatible versions/revisions of a resource or document. Please see the <*procname*> elementary type for a more thorough discussion of version and revisions.

5.2 General Body Comments

%%+ (no keywords)

Any document structuring comment that must be continued on another line to avoid violating the 255-character line length constraint must use the %%+ notation to indicate that a comment line is being continued. This notation may be used after any of the document comment conventions, but may only be necessary in those comments that provide a large list of names, such as %%DocumentResources:. Here is an example of its use:

```
%%DocumentResources: font Palatino-Roman Palatino-Bold
%%+ font Palatino-Italic Palatino-BoldItalic Courier
%%+ font Optima LubalinGraph-DemiOblique
```

See section 3, “DSC Conformance,” for more information about line length and restrictions.

%%BeginBinary: **<***bytecount***>**
<*bytecount*> ::= <*uint*>

%%EndBinary (no keywords)

These comments are used in a manner similar to the %%Begin(End)Data: comments. The %%Begin(End)Binary: comments are designed to allow a document manager to effectively ignore any binary data these comments encapsulate.

To read data directly from the input stream in the PostScript language (using **currentfile**, for instance), it is necessary to invoke a procedure followed immediately by the data to be read. If the data is embedded in the %%Begin(End)Binary: construct, those comments are effectively *part of the data*, which typically is not desired. To avoid this problem, the procedure invocation should fall *inside* the comments, even though it is not binary, and the *bytecount* should reflect this so it can be skipped correctly. In the case of a byte count, allow for carriage returns, if any.

Note *This comment has been included for backward compatibility only and may be discontinued in future versions of the DSC; use the more specific* %%Begin(End)Data: *comments instead.*

%%BeginData: **<***numberof***>[<***type***> [<***bytesorlines***>]]**
<*numberof*> ::= <*uint*> (Lines or physical bytes)
<*type*>::= Hex | Binary | ASCII (Type of data)
<*bytesorlines*> ::= Bytes | Lines (Read in bytes or lines)

%%EndData (no keywords)

These comments are designed to provide information about embedded bodies of data. When a PostScript language document file is being parsed, encountering raw data can tremendously complicate the parsing process. Encapsulating data within these comments can allow a document manager to ignore the enclosed data, and speed the parsing process. If the *type* argument is missing, binary data is assumed. If the *bytesorlines* argument is missing, *numberof* should be considered to indicate bytes of data.

Note that *<numberof>* indicates the bytes of *physical* data, which vary from the bytes of *virtual* data in some cases. With hex, each byte of *virtual* data is represented by two ASCII characters (two bytes of *physical* data). Although the PostScript interpreter ignores white space in hex data, these count toward the byte count.

For example,

```
FD 10 2A 05
```

is 11 bytes of *physical* data (8 bytes hex, 3 spaces) and 4 binary bytes of *virtual* data.

Remember that binary data is especially sensitive to different print environments because it is an 8-bit representation. This can be very important to the document manager if a print network has a channel that is 7 bit serial, for example. See also the %%DocumentData: comment.

To read data directly from the input stream (using **currentfile**, for instance), it is necessary to invoke a procedure followed *immediately* by the data to be read. If the data is embedded in the %%Begin(End)Data: construct, then those comments are effectively *part of the data*, which is typically not desirable. To avoid this problem, the procedure invocation should fall *inside* the comments, even though it is not binary, and the byte or line counts should reflect this so it can be skipped correctly. In the case of a byte count, allow for end-of-line characters, if any.

Note *Document managers should ensure that the entire* %%BeginData: *comment line is read before acting on the byte count.*

In the example below, there are 135174 bytes of hex data, but the %%BeginData: and %%EndData comments encompass the call to the **image** operator. The resulting byte count includes 6 additional bytes, for the string “image” plus the newline character.

```
/picstr 256 string def
25 140 translate
132 132 scale
256 256 8 [256 0 0 -256 0 256] { currentfile picstr readhexstring pop }
%%BeginData: 135174 Hex Bytes
image
4c47494b3187c237d237b137438374ab
213769876c8976985a5c987675875756
...Additional 135102 bytes of hex...
%%EndData
```

Instead of keeping track of byte counts, it is probably easier to keep track of *lines* of data. In the following example, the line count is increased by one to account for the “image” string:

```
/picstr 256 string def
25 140 translate
132 132 scale
256 256 8 [256 0 0 -256 0 256] { currentfile picstr readhexstring pop }
%%BeginData: 4097 Hex Lines
image
4c47494b3187c237d237b137438374ab
213769876c8976985a5c987675875756
...Additional 4094 lines of hex...
%%EndData
```

With binary data, it is unlikely that the concept of lines would be used, because binary data is usually considered one whole stream.

%%BeginDefaults (no keywords)

%%EndDefaults (no keywords)

These comments identify the start and end of the *defaults* section of the document. These comments can only occur after the header section (%%EndComments), after the EPSI preview (%%Begin(End)Preview), if there is one, but before the prolog (%%BeginProlog) definitions.

Some page level comments that are similar to header comments can be used in this defaults section of the file to denote default requirements, resources, or media for all pages. This saves space in large documents (page-level values do not need to be repeated for every page) and can give the document manager some hints on how it might optimize resource usage in the file. The only comments that can be used this way are the following:

%%PageBoundingBox:
%%PageCustomColors:
%%PageMedia:
%%PageOrientation:
%%PageProcessColors:
%%PageRequirements:
%%PageResources:

For example, if the %%PageOrientation: Portrait comment were used in the defaults section, it would indicate that the default orientation for all pages is portrait. When page-level comments are used this way they are known as *page defaults*. Page comments used in a page override any page defaults in effect. In reference to the previous example, if a particular page of the document were to have a landscape orientation, it would place a %%PageOrientation: Landscape comment after the %%Page: comment to override the default portrait orientation.

Example 2 illustrates the page default concept.

Example 2

```
%!PS-Adobe-3.0
%%Title: (Example of page defaults)
%%DocumentNeededResources: font Palatino-Roman Helvetica
%%DocumentMedia: BuffLetter 612 792 75 buff ( )
%%+ BlueLetter 612 792 244 blue (CorpLogo)
%%EndComments
%%BeginDefaults
%%PageResources: font Palatino-Roman
%%PageMedia: BuffLetter
%%EndDefaults
%%BeginProlog
...Prolog definitions...
%%EndProlog
%%BeginSetup
...PostScript language instructions to set the default paper size, weights, and
color...
%%EndSetup
%%Page: Cover 1
%%PageMedia: BlueLetter
%%BeginPageSetup
...PostScript language instructions to set the blue corporate logo cover paper...
%%EndPageSetup
...Rest of page 1...
%Page: ii 2
%%PageResources: font Palatino-Roman Helvetica
...Rest of page 2...
%%Page: iii 3
...Rest of the document...
%%EOF
```

In this example, the font resource Palatino-Roman is specified in the defaults section as a page resource. This indicates that Palatino-Roman is a page default and will most likely be used on every page. Also, the media BuffLetter is specified as the page default. Buff-colored, 20-lb, 8.5" x 11" paper will be used for most pages.

Page 1 uses a special blue cover paper and overrides the page default (buff paper) by putting a %%PageMedia: comment in the page definition. Page 2 uses buff paper and therefore doesn't have to put the %%PageMedia: comment in its page definition. However, it does use the Helvetica font in addition to the Palatino-Roman font. The page default of Palatino-Roman is overridden by the %%PageResources: comment in the page definition.

Note *In some instances it may be superfluous to use these page defaults. If only one type of orientation, media type, etc. is used in the entire document, the header comment alone is sufficient to indicate the default for the document. Page defaults should only be used if there is more than one bounding box, custom color, medium, orientation, process color, requirement, or resource used.*

%%BeginEmulation: **<***mode***>**
<*mode*> ::= diablo630 | fx100 | lj2000 | hpgl | hplj | impress | ti855

%%EndEmulation (no keyword)

The %%BeginEmulation: comment signifies that the input data following the comment contains some printer language other than PostScript. The first line after the %%BeginEmulation comment should be the PostScript language instructions to invoke the emulator. This code is in the PPD file for the printer. Note that the invocation of the emulator is restricted to one line.

This comment enables a document manager to route the document or piece of the document to an appropriate printer. The %%EndEmulation comment should be preceded by the code to switch back to PostScript mode on printers that support this type of switching (again, limit this code to one line). Alternatively, the %%EndEmulation comment may be omitted, in which case the end-of-file switches the printer back into PostScript mode. The following example illustrates the first approach:

```
%!PS-Adobe-3.0
%%Title: (Example of emulator comments)
%%Emulation: hplj
%%EndComments
...Prolog definitions and document setup...
%%BeginEmulation: hplj
3 setsoftwareiomode        % Invoke hplj emulation
...Emulator data...
1B 7F 30                   % Switch back to PostScript
%%EndEmulation
...Remainder of document...
```

Note *When including emulator data, this may break the page independence constraint for a conforming PostScript language file, because there is no way to signify page boundaries. Care should be taken when invoking specialized features of the document manager, such as n-up printing. The document may not be printed as expected.*

%%BeginPreview: **<*width*> <*height*> <*depth*> <*lines*>**

<*width*> ::= <*uint*>	(Width of the preview in pixels)
<*height*> ::= <*uint*>	(Height of the preview in pixels)
<*depth*> ::= <*uint*>	(Number of bits of data per pixel)
<*lines*> ::= <*uint*>	(Number of lines in the preview)

%%EndPreview (no keywords)

These comments bracket the preview section of an EPS file in interchange format (EPSI). The EPSI format is preferred over other platform-dependent previews (for example, Apple Macintosh and IBM PC) when transferring EPS files between heterogenous platforms. The *width* and *height* fields provide the number of image samples (pixels) for the preview. The *depth* field indicates how many bits of data are used to establish one sample pixel of the preview (typical values are 1, 2, 4, or 8). The *lines* field indicates how many lines of hexadecimal data are contained in the preview, so that an application disinterested in the preview can easily skip it.

The preview consists of a bitmap image of the file, as it would be rendered on the page by the printer or PostScript language previewer. Applications that use the EPSI file can use the preview image for on-screen display. Each line of hexadecimal data should begin with a single percent sign. This makes the entire preview section a PostScript language comment so the file can be sent directly to a printer without modification. See section 6, "Device-Independent Screen Preview," of the *Encapsulated PostScript Specifications* available from the Adobe Systems Developers' Association.

The EPSI preview should be placed after the %%EndComments in the document file, but before the defaults section (%%Begin(End)Defaults), if there is one, and before the prolog (%%BeginProlog) definitions.

Note *Preview comments can be used only in documents that comply with the EPS file format. See the* Encapsulated Postscript Specifications *available from the Adobe Systems Developers' Association for more details, including platform-specific versions of the preview (Apple Macintosh and IBM PC platforms).*

%%BeginProlog (no keywords)

%%EndProlog (no keywords)

These comments delimit the beginning and ending of the prolog in the document. The prolog must consist only of procset definitions. The %%EndProlog comment is widely used and parsed for, and must be included in all documents that have a distinct prolog and script.

Breaking a document into a prolog and a script is conceptually important, although not all document descriptions fall neatly into this model. If your document represents free form PostScript language fragments that might entirely be considered a *script*, you should still include the %%EndProlog comment, even though there may be nothing in the prolog part of the file. This effectively makes the entire document a script.

See section 3.1, "Conforming Documents," and 4, "Document Structure Rules," for more information on the contents of the document prolog.

%%BeginSetup (no keywords)

%%EndSetup (no keywords)

These comments delimit the part of the document that does device setup for a particular printer or document. There may be instructions for setting page size, invoking manual feed, establishing a scale factor (or "landscape" mode), downloading a font, or other document-level setup. Expect to see liberal use of the **setpagedevice** operator and **statusdict** operators between these two comments. There may also be some general initialization instructions, such as setting some aspects of the graphics state. This code should be limited to setting those items of the graphics state, such as the current font, transfer function, or halftone screen, that will not be affected by **initgraphics** or **erasepage** (**showpage** performs these two operations implicitly). Special care must be taken to ensure that the document setup code modifies the current graphics state and does not replace it. See Appendix I of the *PostScript Language Reference Manual, Second Edition* for more information about how to properly modify the graphics state.

If present, these comments appear after the %%EndProlog comment, but before the first %%Page: comment. In other words, these comments are not part of the prolog. They should be in the first part of the script before any pages are specified.

5.3 General Page Comments

Some of the following general page comments that specify the bounding box or orientation may appear in the defaults section or in a particular page. If these comments appear in the defaults section of the document file between %%BeginDefaults and %%EndDefaults, they are in effect for the entire print job. If they are found in the page-level comments for a page, they should be in effect only for that page. See %%Begin(End)Defaults for more details on page defaults.

%%BeginObject: **<name> [<code>]**
<*name*> ::= <*text*> (Name of object)
<*code*> ::= <*text*> (Processing code)

%%EndObject (no keywords)

These comments delimit individual graphic elements of a page. In a context where it is desirable to be able to recognize individual page elements, this comment provides a mechanism to label and recognize them at the PostScript language level. Labelling is especially useful when a document printing system can print selected objects in a document or on a page.

For instance, the *code* field of this comment can be used to represent *proofing levels* for a document. For example, the printing manager may be requested to “print only those objects with proofing levels less than 4.” This can save printing time when proofing various elements of a document. It can also be useful in systems that allow PostScript language program segments to be parsed and re-edited into convenient groupings and categorizations of graphic page elements. In a document production system or in an application that is highly object-oriented, use of this comment is strongly recommended.

The user must specify to the application what things constitute an object and what the proofing level of each object will be.

%%BeginPageSetup (no keywords)

%%EndPageSetup (no keywords)

These comments are analogous to the %%BeginSetup: and %%EndSetup comments, except that %%BeginPageSetup: and %%EndPageSetup appear in the body of a document right after a %%Page: comment. They delimit areas that set manual feed, establish margins, set orientation, download fonts or other resources for the page, invoke particular paper colors, and so on. This is the proper place to set up the graphics state for the page. It should be assumed that an **initgraphics** and an **erasepage** (i.e. **showpage**) have been performed prior to this page. Take special care to ensure that the code in the page setup *modifies* the current graphics state rather than replaces it. See Appendix I of the *PostScript Language Reference Manual, Second Edition* for more information about how to properly modify the graphics state.

%%Page: <label> <ordinal>

<*label*>::= <*text*> (Page name)
<*ordinal*> ::= <*uint*> (Page position)

This comment marks the beginning of the PostScript language instructions that describe a particular page. %%Page: requires two arguments: a *page label* and a *sequential page number*. The label may be anything, but the ordinal page number must reflect the position of that page in the body of the PostScript language file and must start with 1, not 0. In the following example, the name of the third page of the document is 1:

```
%!PS-Adobe-3.0
...Document prolog and setup...
%%Page: cover 1
...Rest of the cover page...
%%Page: ii 2
...Rest of the ii page...
%%Page: 1 3
...Rest of the first page...
%%Page: 2 4
...Rest of the second page...
%%EOF
```

A document manager should be able to *rearrange* the contents of the print file into a different order based on the %%Page: comment (or the pages may be printed in parallel, if desired). The %%PageOrder: Special comment can be used to inform a document manager that page reordering *should not* take place.

%%PageBoundingBox: { <llx> <lly> <urx> <ury> } | (atend)

<*llx*> ::= <*int*> (Lower-left x coordinate)
<*lly*> ::= <*int*> (Lower-left y coordinate)
<*urx*> ::= <*int*> (Upper-right x coordinate)
<*ury*> ::= <*int*> (Upper-right y coordinate)

This comment specifies the bounding box that encloses all the marks painted on a particular page (this is *not* the bounding box of the whole document—see the %%BoundingBox: comment). *llx*, *lly* and *urx*, *ury* are the coordinates of the lower-left and upper-right corners of the bounding box in the *default user coordinate system* (PostScript units). This comment can pertain to an individual page or a document, depending on the location of the comment. For example, the comment may be in the page itself or in the document defaults section.

%%PageOrientation: Portrait | Landscape

This comment indicates the orientation of the page and can be used by preview applications and post-processors to determine how to orient the viewing window. This comment can pertain to an individual page or a document, depending on the location of the comment. For example, the comment may be in the page itself or in the document defaults section. See %%Orientation: for a description of the various orientations. See %%Begin(End)Defaults for use of this comment as a page default.

5.4 General Trailer Comments

Some trailer comments are special and work with other comments that support the (atend) notation. In addition, trailer comments delimit sections of PostScript language instructions that deal with cleanup and other housekeeping. This cleanup can affect a particular page or the document as a whole.

%%PageTrailer (no keywords)

This comment marks the end of a page. Any page comments that may have been deferred by the (atend) convention should follow the %%PageTrailer comment.

%%Trailer (no keywords)

This comment must only occur once at the end of the document *script*. Any post-processing or cleanup should be contained in the *trailer* of the document, which is anything that follows the %%Trailer comment. Any of the document-level structure comments that were *deferred* by using the (atend) convention must be mentioned in the trailer of the document after the %%Trailer comment.

When entire documents are embedded in another document file, there may be more than one %%Trailer comment as a result. To avoid ambiguity, embedded documents must be delimited by the %%BeginDocument: and %%EndDocument comments.

%%EOF (no keywords)

This comment signifies the end of the document. When the document manager sees this comment, it issues an end-of-file signal to the PostScript interpreter. This is done so system-dependent file endings, such as Control-D and end-of-file packets, do not confuse the PostScript interpreter.

6 Requirement Conventions

The requirement conventions are comments that suggest document manager action. Some of these comments list the resources needed or supplied by the document, delimit those resources if they are supplied, and specify the insertion point for those resources if they are needed. Other comments deal with printer-specific features (listing requirements, delimiting portions of and indicating insertion points for printer specific code) and are used in tandem with the **setpagedevice** operators or **statusdict** operators, as well as the PostScript printer description (PPD) files.

Note *Use of the* %%Include *or* %%Operator *comments in an environment that does not have a document manager can result in the document being processed incorrectly.*

6.1 Requirement Header Comments

%%DocumentMedia: <medianame> <attributes>

<*medianame*> ::= <*text*>	(Tag name of the media)
<*attributes*>::= <*width*> <*height*> <*weight*> <*color*> <*type*>	
<*width*> ::= <*real*>	(Width in PostScript units)
<*height*> ::= <*real*>	(Height in PostScript units)
<*weight*> ::= <*real*>	(Weight in g/m^2)
<*color*> ::= <*text*>	(Paper color)
<*type*> ::= <*text*>	(Type of pre-printed form)

This comment indicates all types of paper media (paper sizes, weight, color) this document requires. If any of the attributes are not applicable to a particular printing situation, zeroes must be substituted for numeric parameters and null strings must be substituted for text parameters. Each different medium that is needed should be listed in its approximate order of *descending* quantity used.

```
%%DocumentMedia: Plain 612 792 75 white ( )
%%+ BlueCL 612 792 244 blue CorpLogo
%%+ Tax 612 792 75 ( ) (1040)
```

The preceding example indicates that the following media are needed for this job:

- 8.5" x 11", 20 lb. paper (Bond lbs × 3.76 = g/m^2).
- Cover pages in blue 8.5" x 11", 65 lb. paper preprinted with the corporate logo.
- Preprinted IRS 1040 tax forms.

Note that the *type* attribute refers to preprinted forms only, and does *not* refer to the PostScript language concept of form objects as resources. The following keywords for the *type* name are defined for general use:

19HoleCerlox	ColorTransparency	CustLetterHead	Tabs
3Hole	CorpLetterHead	DeptLetterHead	Transparency
2Hole	CorpLogo	Labels	UserLetterHead

The related %%PageMedia: comment explicitly calls for the medium that each page requires by referring to its *medianame*.

%%DocumentNeededResources: <resources> | (atend)

This comment provides a list of resources the document needs—that is, resources *not* contained in the document file. This comment is intended to help a document manager decide whether further parsing of the document file is necessary to provide these needed resources. There must be at least one corresponding instance of the %%IncludeResource: comment for each resource this comment lists.

The application that produces the print file must not make any assumptions about which resources are resident in the output device; it must list all resources the document needs. Even if it is a resource, such as the Times-Roman font program, that exists in nearly all implementations, it must appear here. A resource must not be listed if it is not used anywhere in the document.

As a general rule, different types of resources should be listed on separate lines using the %%+ comment, as illustrated in the following example:

```
%%DocumentNeededResources: font Times-Roman Helvetica StoneSerif
%%+ font Adobe-Garamond Palatino-Roman
%%+ file /usr/lib/PostScript/logo.ps
%%+ procset Adobe_Illustrator_abbrev 1.0 0
%%+ pattern hatch bubbles
%%+ form (corporate order form)
%%+ encoding JIS
```

%%DocumentSuppliedResources: **<resources> | (atend)**

The %%DocumentSuppliedResources: comment contains extra information for document managers designed to store and reuse the resources, and provides helpful directories of the resources contained in the print file. This comment lists all resources that have been *provided* in the document print file. There is a %%BeginResource: and %%EndResource pair for each resource in this list. It is assumed that all resources on the %%DocumentSuppliedResources: list are mutually exclusive of those resources found on the %%DocumentNeededResources: list.

%%DocumentPrinterRequired: <print> <prod> [<vers> [<rev>]]

<*print*> ::= <*text*>	(Printer name and print zone)
<*prod*> ::= <*text*>	(Product string or nickname)
<*vers*> ::= <*real*>	(Version number)
<*rev*> ::= <*uint*>	(Revision number)

This comment indicates that the PostScript language instructions in the document are intended for a particular printer, which is identified by its network printer name, nickname, or product string. The printer can optionally be identified by its version and revision strings, as defined by the printer's PPD file or as returned by the **product**, **version**, and **revision** operators.

%%DocumentPrinterRequired: can be used to request a particular printer in a highly networked environment where that printer may be more convenient or to override document manager defaults and prevent re-routing of the document. It can also be used if the PostScript language file itself contains printer-specific elements. This last case should rarely be necessary, as most documents requiring particular features of a PostScript printer can provide requirement conventions indicating a need for that feature, rather than require a particular printer. Then, if other printers are available that have the necessary features, the document may still be printed as desired. The following example unconditionally routes the document to a printer called SEVILLE in the network's "Sys_Marketing" zone:

```
%%DocumentPrinterRequired: (SEVILLE@Sys_Marketing) ( )
```

If the nickname of the printer is used (this is often necessary to differentiate among different models of printers), the version/revision numbers that are part of the nickname should be ignored.

For example, the product name for a series of printers may be (SpeedyLaser). There are several models of SpeedyLaser printers, the SL300, SL600, and SL1200. The nicknames of these printers are (SL300 Version 47.2), (SL600 Version 48.1), and (SL1200 Version 49.4). To specify the need for a SL600 printer, the nickname (excluding the version number) should be used. For example:

```
%%DocumentPrinterRequired: ( ) (SL600)
```

The version and revision numbers in this comment should be used infrequently.

%%DocumentNeededFiles: { <filename> ... } | (atend)

The comment %%DocumentNeededFiles: lists the files a document description needs. Each file mentioned in this list appears later in the document as the argument of an %%IncludeFile: comment. It is assumed that files on the %%DocumentNeededFiles: list do not include those appearing on the %%DocumentSuppliedFiles: file list.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general comment* %%DocumentNeededResources: *instead.*

%%DocumentSuppliedFiles: { <filename> ... } | (atend)

The comment %%DocumentSuppliedFiles: lists the files in a document description. Each file mentioned in this list appears later in the document in the context of a %%BeginFile: and %%EndFile: comment construct. It is assumed that files on the %%DocumentSuppliedFiles: list do not include those appearing on the %%DocumentNeededFiles: file list.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general comment* %%DocumentSuppliedResources: *instead.*

%%DocumentFonts: { <fontname> ... } | (atend)

This comment indicates that the print job uses all fonts listed. In particular, there is at least one invocation of the **findfont** or **findresource** operator for each of the font names listed. The application producing the print file should not make any assumptions about which fonts are resident in the printer (for example, Times-Roman). Note that the list of font names for %%DocumentFonts: should be the union of the %%DocumentNeededFonts: and %%DocumentSuppliedFonts: font lists. If the list of font names exceeds the 255 characters-per-line limit, the %%+ comment should be used to extend the line.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general comments* %%DocumentNeededResources: *and* %%DocumentSuppliedResources: *instead.*

%%DocumentNeededFonts: { <fontname> ... } | (atend)

This comment provides a list of fonts the document *requires* and are *not* contained in the document file. It is assumed that fonts on the %%DocumentNeededFonts: list do not appear on the %%Document-SuppliedFonts: font list. It is also assumed that there is at least one corresponding instance of the %%IncludeFont: comment for each font listed in this section.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general comment* %%DocumentNeededResources: *instead.*

%%DocumentSuppliedFonts: { <fontname> ... } | (atend)

This comment provides a list of font files that have been provided in the document print file as downloaded fonts. It is assumed that fonts on the %%DocumentSuppliedFonts: list do not appear on the %%DocumentNeededFonts: font list. There is at least one corresponding %%BeginFont: and %%EndFont pair in the document description for each of the listed font names.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general comment* %%DocumentSuppliedResources: *instead.*

%%DocumentProcSets: { <procname> ... } | (atend)

This comment provides a list of *all* procsets referenced in the document. Its use is similar to the %%DocumentFonts: comment. The list of procsets for %%DocumentProcSets: should be the union of the %%DocumentNeededProcSets: and %%DocumentSuppliedProcSets: procset lists. If the list of procset names exceeds the 255 characters-per-line limit, the %%+ comment should be used to extend the line.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%DocumentNeededResources: *and* %%DocumentSuppliedResources: *comments instead.*

%%DocumentNeededProcSets: { <procname> ... } | (atend)

This comment indicates that the document needs the listed procsets. It is assumed that procsets on the %%DocumentNeededProcSets: list do not appear on the %%DocumentSuppliedProcSets: procset list. This comment is used whenever any %%IncludeProcSet: comments appear in the file.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general comment* %%DocumentNeededResources: *instead.*

%%DocumentSuppliedProcSets: { <procname> ... } | (atend)

This comment indicates that the document contains the listed procsets. It is assumed that procsets in the %%DocumentSuppliedProcSets: list do not include those appearing on the %%DocumentNeededProcSets: procset list. This comment is used whenever any %%BeginProcSet and %%EndProcSet comments appear within the document.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general comment* %%DocumentSuppliedResources: *instead.*

%%OperatorIntervention: **[<password>]**
<*password*> ::= <*textline*>

This comment causes the document manager to block a print job in the print queue until the printer operator releases the print job for printing. The comment may contain an optional *password* that the print operator must supply to release the job. This allows the printing of sensitive documents to be delayed until the intended recipient is present at the printer to pick up the document.

%%OperatorMessage: **<***textline***>**

If the output device has an appropriate user interface, the %%OperatorMessage: comment provides a message that the document manager can display on the console before printing the job. This comment must only appear in the header of the file.

%%ProofMode: **<mode>**
<*mode*> ::= TrustMe | Substitute | NotifyMe

This comment provides information about the level of accuracy that is required for printing. It is intended to provide guidance to the document manager for appropriate tactics to use when error conditions arise or when resource and feature shortages are encountered.

The three modes may be thought of as instructions to the document manager. If the document manager detects a resource or feature shortage, such as a missing font or unavailable paper size, it should take action based on these proof modes:

- TrustMe—Indicates the document manager should *not* take special action. The intent is that the document formatting programs or the user knows more than the document manager. For example, fonts may be available on a network font server that the document manager does not know about.

  Even with a comment like %%IncludeResource:, if the %%ProofMode is TrustMe, the printing manager should proceed even if a resource cannot be found. The assumption is that the document can compensate for the resource not being included.

- Substitute—Indicates the printing manager should do the best it can to supply missing resources with alternatives. This may mean substituting fonts, scaling pages (or tiling) when paper sizes are not available, and so on. This is the default proofing level and should be used if the *mode* is missing from the comment or if the comment is missing from the document.

- NotifyMe—Indicates the document should not be printed if there are any mismatches or resource shortages noted by the printing manager. For example, when printing on an expensive color printer, if the correct font is not available, the user probably does *not* want a default font. The document manager, if it cancels the print job, should notify the user in some system-specific manner.

These modes are intended for the printing manager to consider *before* it prints the file, based on its own knowledge and queries of available fonts, paper sizes, and other resources. If the file is printed, and an error occurs, that is a separate issue.

%%Requirements: <requirement> [(<style> ...)] ...

<*requirement*> ::= collate | color | duplex | faceup | fax | fold | jog | manualfeed | numcopies | punch | resolution | rollfed | staple

<*style*> ::= <*text*>

This comment describes document requirements, such as duplex printing, hole punching, collating, or other physical document processing needs. These requirements may be activated by the document using **statusdict** operators or **setpagedevice**, or they may be requested using the %%IncludeFeature: comment.

The *requirement* parameter should correspond to a specific printer feature. The optional *style* parameter can be used to further describe the specifics of the processing. For example, the punch requirement has a style to indicate that a printer capable of 19 Hole Cerlox punching is required: punch(19). If more than one style of requirement is necessary, the styles can be listed in the enclosing parentheses (separated by commas) for that requirement. For example, if both positional stapling (staple in the lower right hand corner) and staple orientation (staple at 45 degrees) is desired, the requirement is: staple(*position*,*orient*). This informs the document manager that the printer printing this document must be equipped with a stapler that can position *and* orient the staple.

The %%Requirements: comment can be used to determine if the printer the user selects can meet the document's requirements. If it cannot, the document should be rerouted to a printer that can, otherwise the document is not processed as expected. It is the document manager's responsibility to determine if the printer can fulfill the requirements and if the operator and/or application should be notified of any incapability. See also the %%ProofMode: comment for actions to take when there are no printers available that satisfy the requirements.

Note *The* %%Requirements: *comment is informational only; it does not suggest that the document manager actuate these requirements—that is, turn them on. The PostScript language instructions in the document activate these features.*

The following keywords for the *requirement* parameter are defined:

- collate—Indicates that the document contains code that will instruct the printer to produce collated copies (for example, 1-2-3-1-2-3-1-2-3), rather than uncollated copies (for example, 1-1-1-2-2-2-3-3-3). If collate is not specified, then non-collation of the document should be assumed, except if the duplex, fold, jog, or staple requirements are specified (they imply collation by definition). This requirement should be used in conjunction with the numcopies requirement.

- color—Indicates that the printer must be able to print in color. If this option is not specified, monochrome printing is assumed to be sufficient.

- color(separation)—Indicates that the printer must be able to perform internal color separation. If this style modifier is not specified, composite color output is assumed to be sufficient.

- duplex—Indicates that the document issues commands such that pages are printed on both sides of the paper. Any printer intended to print such a document properly must be capable of producing duplex output.

- duplex(tumble)—Indicates a style of duplex printing in which the logical top of the back side is rotated 180 degrees from the logical top of the front side. A wall calendar is an example of a document that is typically tumble duplexed.

- faceup—Indicates that output pages are stacked face-up. If this requirement is not specified, then the selected printer need not be capable of stacking pages face-up.

- fax—Indicates that the document contains segments of PostScript code pertaining to fax devices and should be sent to a fax-capable printer.

- fold—Indicates that the document requests that the printer fold the resulting output. Typical style modifiers to this requirement would be letter, z-fold, doublegate, leftgate, rightgate, and saddle. These are illustrated in Figure 3.

- jog—Indicates that jobs or multiple replications of the same document are offset-stacked from one another in the output tray. The document manager must ensure that the selected printer has the ability to offset stack job output.

- manualfeed—Indicates that the document requests that paper be fed in from the manual feed slot. If this requirement is not specified, the selected printer need not have a manual feed slot.

- numcopies(<*uint*>)—Indicates that the document instructs the printer to produce <*uint*> number of copies of the output. If this requirement is not specified, a default of numcopies(1) should be assumed.

- punch—Indicates that the document specifies commands concerning hole punching. If punch is not specified, the printer need not be capable of punching.

- punch(<*uint*>)—Indicates that the document contains PostScript language instructions that cause the output to be punched with <*uint*> number of holes. Typical values are 3-, 5-, and 19-hole (Cerlox) punching. If there is no *style* modifier to the punch requirement, 3-hole punching should be assumed to be acceptable.

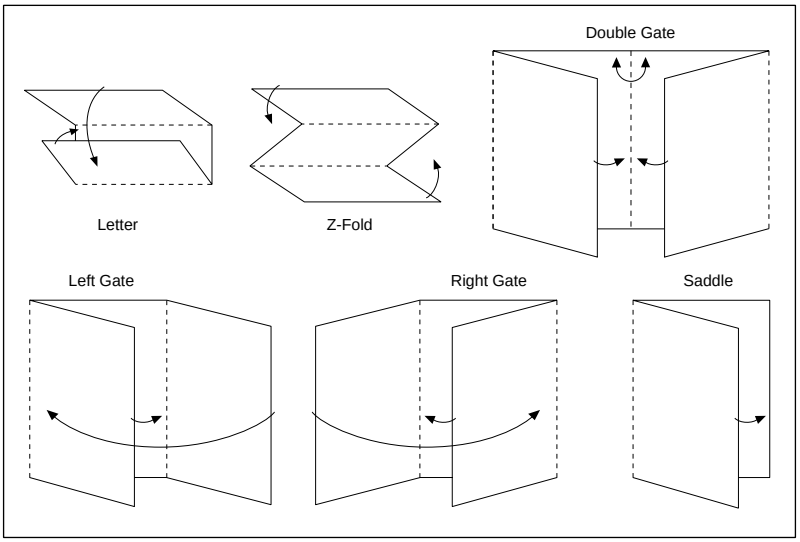
- resolution(*x*, *y*)—Indicates that the printer is set to a particular resolution in the *x* and *y* directions. The printer manager must provide a printer that can print in that resolution. If this requirement is not specified, any printer resolution is acceptable.

- rollfed—Indicates that the document issues commands specific to roll-fed devices, such as where and when to cut the paper, how far to advance the paper, and so on. If this requirement is not specified, the printer need not support roll-fed paper.

- staple—Indicates that PostScript language commands in the document cause the output to be stapled. If staple is not specified as a requirement, the printer need not support stapling.

- staple([*position*],[*orient*])—Indicates a staple position and a staple orientation. A stapler may be able to position staples on a page in several different locations. If the print job needs a printer stapler that performs positioning, this should be indicated by the style keyword *position*. If staple orientation is needed (for example, 0, 45, 90, or 135 degrees), the *orient* style should be included with the staple requirement. If no style modifiers are given, then simple stapling is assumed to be sufficient (top left-hand corner).

Figure 3 *Various fold options*



The order of the arguments to the %%Requirements: comment is significant and implies the order in which the operations occur in the PostScript language code.

Example 3 shows the proper use of the %%Requirements: comment and the associated %%Begin(End)Feature: comments. Three copies of this document will be printed duplex; the copies will be offset in the output tray from one another.

Example 3

```
%!PS-Adobe-3.0
%%Title: (Example of requirements)
%%LanguageLevel: 2
%%Requirements: duplex numcopies(3) jog
%%EndComments
%%BeginProlog
...Various prolog definitions...
%%EndProlog
%%BeginSetup
% For Level 1 this could have been a series of statusdict operators
%%BeginFeature: *Duplex True
<< /Duplex true >> setpagedevice
%%EndFeature
/#copies 3 def
%%BeginFeature: *Jog 3
<< /Jog 3 >> setpagedevice
%%EndFeature
%%EndSetup
...Rest of the document...
%%EOF
```

Note that in this instance, calls to **setpagedevice** are separated for each feature. This enables a document manager to re-route the document to a Level 1 printer. If output is going to a Level 2 printer only, the following could have been used:

```
<< /Duplex true  /NumCopies true  /Jog 3 >>
setpagedevice
```

Because Level 2 feature activation is device independent, the %%Begin(End)Feature: comments are unnecessary if the document is confined to Level 2 interpreters. The %%Requirements: and the %%LanguageLevel: comments are still necessary, however.

Note *This comment lists all of the requirements for a particular job; individual pages may use some of the requirements in different combinations. To specify what the page requirements are for a particular page or for the whole document (page defaults), see the* %%PageRequirements: *comment.*

%%VMlocation: global | local

This comment is to inform resource users if a resource can be loaded into global or local VM. For all resource categories other than a font, the operator **findresource** unconditionally executes **true setglobal** before executing the file that defines the resource. This means a resource is loaded into global VM unless **false setglobal** appears in the resource definition.

The creator of a resource must determine if the resource works correctly in global VM. If it does, the resource must not execute **setglobal**. The resource may wish to include the %%VMlocation: global comment. The resource is loaded into global VM by **findresource**, but will be loaded into current VM under the control of a document manager if it is explicitly downloaded.

If the resource does not work in global VM or if the creator of the resource does not know if the resource will work reliably in global VM, the resource must use the %%VMlocation: local comment and the following PostScript language fragment:

```
currentglobal
false setglobal
...Definition of the resource, including
defineresource...
setglobal
```

%%VMusage: **<***max***> <min>**

<*max*> ::= <*uint*> (Maximum VM used by resource)<*min*> ::= <*uint*> (Minimum VM used by resource)

The document manager can use the information supplied by this comment to determine if the PostScript language interpreter has enough VM storage to handle this particular resource. This comment should be used only in static resource files, such as fonts, procsets, files, forms, and patterns, which are all resources that rarely change and should not generally be used in page descriptions.

max indicates the amount of VM storage this resource consumes if it is the first resource of its type to be downloaded. *min* indicates the minimum amount of VM this resource needs. The numbers may not be equal because some resources, such as fonts, can share VM storage in some versions of the PostScript interpreter. In synthetic fonts, for example, the **charstrings** of the font may be shared.

These numbers are not determined in the resource. Rather, they are determined by the resource creator when the resource (for example, a font) is initially programmed. The numbers are placed in the resource as static entities in this comment. To achieve accurate results when determining the *usage* values, make sure there are no dependencies on other resources or conditions.

The VM a resource uses can be found by issuing the **vmstatus** command before and after downloading a resource, and then again after downloading the same resource a second time. The difference between the first and second numbers (before and after the first downloading) yields the *max* value; the difference between the second and third (after the second download) yields the *min* value. The following example illustrates how to obtain the *max* and *min* values for a resource:

```
vmstatus pop /vmstart exch def pop
...The resource goes here...
vmstatus pop dup vmstart sub (Max: ) print == flush
/vmstart exch def pop
...The resource goes here...
vmstatus pop vmstart sub (Min: ) print == flush pop
```

Note *To obtain accurate memory usage values, it is important to turn off the garbage collection mechanism in Level 2.*

6.2 Requirement Body Comments

Some of the comments listed in this section, if used, must have a corresponding comment in the header of the document. For example, if the %%IncludeResource: comment is used, there must be a %%DocumentNeededResources: comment in the header of the document.

Table 2 *Body and header comment usage*

Body Comment Used	*Corresponding Header Comment*
%%Begin(End)Document:	%%DocumentSuppliedResources: file
%%IncludeDocument:	%%DocumentNeededResources: file
%%Begin(End)Resource:	%%DocumentSuppliedResources:
%%IncludeResource:	%%DocumentNeededResources:
%%Begin(End)File:	%%DocumentSuppliedResources: file
%%IncludeFile:	%%DocumentNeededResources: file
%%Begin(End)Font:	%%DocumentSuppliedResources: font
%%IncludeFont:	%%DocumentNeededResources: font
%%Begin(End)ProcSet:	%%DocumentSuppliedResources: procset
%%IncludeProcSet:	%%DocumentNeededResources: procset
%%Begin(End)Feature:	%%Requirements: or %%DocumentMedia:
%%IncludeFeature:	%%Requirements: or %%DocumentMedia

%%Begin and %%End comments indicate that the PostScript language instructions enclosed by these comments is a resource, feature, or document. An intelligent document manager may save resources for future use by creating a resource library on the host system. The document manager may replace printer-specific feature instructions when rerouting the document to a different printer, or may ignore duplicate DSC comments in an included document. The proper use of these comments facilitates this intelligent document handling.

%%Include comments indicate that the named resource, feature, or document (for example, font, procset, file, paper attribute, EPS file, and so on) should be included in the document at the point where the comment is encountered. The document manager fulfills these requirements so there is an inherent risk in using these comments in a document. If there is no document manager in your system environment, the document may not print correctly. As the DSC become more prevalent and strictly adhered to, there will be more document manager products available to take advantage of these %%Include comments.

%%BeginDocument: **<name> [<version> [<type>]]**

<*name*> ::= <*text*>	(Document name)
<*version*> ::= <*real*>	(Document version)
<*type*> ::= <*text*>	(Document type)

%%EndDocument (no keywords)

These comments delimit an *entire conforming document* that is imported as part of another PostScript language document or print job. The *name* of the document is usually environment-specific; it can be an operating system file name or a key to a document database. The *version* and *type* fields are optional and, if used, should provide extra information for recognizing specific documents (an example of usage is a version control system).

The %%BeginDocument: comment is necessary to allow multiple occurrences of the %!PS-Adobe-3.0, %%EndProlog, %%Trailer, and %%EOF comments in the body of a document. Any document file that is embedded within another document file *must* be surrounded by these comments.

Note *All feature and resource requirements of an included (child) document should be inherited by the including (parent) document. For example, if a child document needs the StoneSerif font resource, this must be reflected in the* %%DocumentNeededResources: *comment of the parent. This is necessary so document managers can examine the top level header of any document and know all resources and features that are required.*

%%IncludeDocument: **<***name***> [<***version***> [<***revision***>]]**

<*name*> ::= <*text*>	(Document name)
<*version*> ::= <*real*>	(Version of the document)
<*revision*> ::= <*int*>	(Revision of version)

This comment is much like the %%IncludeResource: file comment except that it specifies that the included file is a *conforming document description* rather than a small portion of stand-alone PostScript language code. This means that, in all probability, the document contains at least one instance of **showpage**, and the included document should be wrapped with a **save** and **restore**. In particular, illustrations and EPSF files that have no effect other than to make marks on a page are perfectly suited for the %%IncludeDocument: convention.

When a document file is printed, usually a certain amount of PostScript language code is added to the file. Such code may deal with font downloading issues, paper sizes, or other aspects of printing once a printer has been selected for the document. At that stage, the printing manager must remove the %%IncludeDocument: comment and embed the requested document (along with all the structuring conventions that may fall within that file) between %%BeginDocument: and %%EndDocument comments.

%%BeginFeature: **<***featuretype***> [<***option***>]**
<*featuretype*> ::= <*text*> (PPD feature name)
<*option*> ::= <*text*> (Feature option)

%%EndFeature (no keywords)

The %%BeginFeature and %%EndFeature comments delimit any PostScript language fragments that invoke a printer-specific feature on a printer. The *featuretype* corresponds to one of the keywords in the PostScript printer description (PPD) file, and the *featuretype option* sequence must be exactly as it is found in the PPD file so it cooperates effectively with these conventions.

A document manager may choose to replace the enclosed PostScript language code with the proper sequence of instructions if the document is sent to a different printer than originally intended. In a sense, this is the opposite of the %%IncludeFeature: comment, which indicates that the document manager must invoke the specified printer feature at that position in the print file. The next two examples set up an imageable region for a job. Example 4 uses the Level 1 **statusdict** method of selecting page size. Example 5 uses the new Level 2 **setpagedevice** operator.

Example 4

```
%%BeginFeature: *PageSize Legal
legal
%%EndFeature
```

Example 5

```
%%BeginFeature: *PageSize Legal
 << /PageSize [612 1004] >> setpagedevice
%%EndFeature
```

%%IncludeFeature: **<featuretype> [<option>]**
<*featuretype*> ::= <*text*> (Name of desired feature)
<*option*> ::= <*text*> (Feature option)

This comment specifies the need for a particular printer feature, as described in the PostScript printer description (PPD) file. Its use specifies a *requirement* a document manager must fulfill before printing (see also the discussion under %%BeginFeature). The document file may make the assumption that the %%IncludeFeature line in the file is replaced by the appropriate PostScript language fragment from the appropriate PPD file, and that the execution of the file may be contextually dependent upon this replacement. This offers a very powerful way of making a document behave differently on different printers in a device-independent manner. See the *PostScript Printer Description Files Specification* for more information about PPD files.

%%BeginFile: <filename>

%%EndFile (no keywords)

The enclosed segment is a fragment of PostScript language code or some other type of resource that does not fall within any of the other resource categories. The file-server component of a document manager may extract a copy of this file for later use by the %%IncludeFile: or %%IncludeResource: file comments. The file name will usually correspond to the original disk file name on the host system.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%Begin(End)Resource: *comments instead.*

%%IncludeFile: <filename>

Indicates that the document manager must insert the specified file at the current position in the document. The file name specified also must appear in the %%DocumentNeededResources: file or the %%DocumentNeededFiles: list.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%IncludeResource: *comment instead.*

%%BeginFont: <fontname> [<printername>]
<*printername*> ::= <*text*>

%%EndFont (no keywords)

These comments delimit a downloaded font. The font-server component of a document manager may remove the font from the print file (for instance, if the font is already resident on the chosen printer) or it may simply keep a copy of it for later use by the %%IncludeFont: or %%IncludeResource: font comments. The *fontname* field must be the valid PostScript language name of the font as used by the **definefont** operator, and the optional *printername* field may contain the network name of the printer, in an environment where fonts may be tied to particular printers.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%Begin(End)Resource: *comments instead.*

%%IncludeFont: <fontname>

Indicates that the document manager must include the specified font at the current position in the document. The *fontname* specified should be the correct PostScript language name for the font (without the leading slash). Due to the presence of multiple **save/restore** contexts, a document manager may have to supply a specific font more than once in one document, and should do so whenever this comment is encountered.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%IncludeResource: *comment instead.*

%%BeginProcSet: <procname>

%%EndProcSet (no keywords)

The PostScript language instructions enclosed by the %%BeginProcSet: and %%EndProcSet comments typically represents some subset of the document prolog. The prolog may be broken down into many subpackages, or procedure sets (procsets), which may define groups of routines appropriate for different imaging requirements. These individual procsets are identified by name, version, and revision numbers for reference by a document management system. A document manager may choose to extract these procsets from the print file to manage them separately for a whole family of documents. An entire document prolog may be an instance of a procset, in that it is a body of procedure definitions used by a document description file. (See the %%DocumentProcSets:, %%IncludeProcSet:, and %%IncludeResource: *procset* comments). The *name*, *version*, and *revision* fields should uniquely identify the procset. The *name* may consist of a disk file name or it may use a PostScript language name under which the prolog is stored in the printer. See the %%?Begin(End)ProcSetQuery: and the %%?Begin(End)ResourceQuery: *procset* comment, which one may use to query the printer or document manager for the prolog name and version fields.

A document manager may assume that the document prolog consists of everything from the beginning of the print file through the %%EndProlog comment, which may encompass several instances of the %Begin(End)ProcSet: comments.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%Begin(End)Resource: *comments instead.*

%%IncludeProcSet: <procname>

This is a special case of the more general %%IncludeResource: file comment. It requires that a PostScript language procset with the given name, version, and revision be inserted into the document at the current position. If a version-numbering scheme is not used, these fields should still be filled with a "dummy" value, such as 0. See the %%Begin(End)Resource: and %DocumentNeededResources: comments.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%IncludeResource: *comment instead.*

%%BeginResource: <resource> [<max> <min>]

<*max*> ::= <*uint*> (Maximum VM used by resource)
<*min*> ::= <*uint*> (Minimum VM used by resource)

%%EndResource (no keywords)

These comments delimit a resource that is defined by PostScript language code directly in the document file—for example, downloadable fonts. The resource-management component of the document manager may remove the resource from the print file and replace it with an %%IncludeResource comment (for instance, if the chosen printer already has the resource resident) or it may simply keep a copy of it for later use by the %%IncludeResource: comment. The resource name specified should also appear in the %%DocumentSuppliedResources: list.

The optional *usage* parameters should be supplied if the %%VMusage: comment is not provided in the resource. A document manager can use these numbers to determine if a particular resource will fit inside the printer VM. If it cannot, the document manager may move the resource within the print file, juggling resources until the file can fit, or it may reroute the print file to a printer with more VM. See the %%VMusage: comment for details on how to obtain these numbers for a resource.

Font note—These comments delimit a font that is being downloaded. The font server component of a document manager may remove the font from the print file (for instance, if the chosen printer already has the font resident) or it may simply keep a copy of it for later use by the %%IncludeResource: comment.

File note—The enclosed segment is a fragment of PostScript language code or some other item that does not fall within the other resource categories. The file-server component of the document manager may extract a copy of this file for later use by the %%IncludeResource: comment. The file name will usually correspond to the original disk file name on the host system.

Procset note—The PostScript language code enclosed by these comments typically represents some subset of the document prolog. The prolog may be broken down into many procedure sets, which may define groups of routines appropriate for different imaging requirements. These individual procsets are identified by a *name*, *version*, and optional *revision* numbers for reference by a print management system. A document manager may choose to extract these procsets from a print file to manage them separately for a whole family of documents. An entire document prolog may be an instance of a procset, in that it is a body of procedure definitions used by a document description file.

%%IncludeResource: <resource>

Indicates that the document manager must include the named resource at this point in the document. The resource name specified also must appear in the %%DocumentNeededResources: list. It is up to the application creating the document to manage memory for resources that employ this comment (using **save/restore** pairs). Although the font example below is specific to fonts, memory management and resource optimization are also applicable to forms, patterns, and other memory-intensive resources.

Font note—In the case of commonly available fonts, it is highly likely that the font server or document manager would ignore the inclusion request, because the fonts would already be available on the printer. However, the %%IncludeResource: font comment must still be included so that if a standard font is not available it can be supplied (there are printers that do not have the 13 standard fonts that are resident in most of Adobe's PostScript implementations). %%IncludeResource: font comments of this nature should be placed in the document setup section.

Due to the presence of multiple **save/restore** contexts, a font server may have to supply a specific font more than once within a single document, and should do so whenever this comment is encountered. Depending on the memory available in the target printer, a document manager may optimize font usage by moving the inclusion of fonts within the document. A frequently used font could be downloaded during the document setup, thus making it available for use by any page. A font that is used on one or two particular pages, could be downloaded during the page setups for each of the individual pages. A special font that is used for one or two paragraphs on one page only would not be moved.

In Example 6, four different fonts (ITC Stone®, Palatino*, Carta®, and Sonata®) are downloaded. The memory management scheme used by the application that generated this code assumes that up to three fonts may be downloaded at any one point in time. Note the use of multiple %%IncludeResource: font comments for the same font when a **save**-**restore** pair "undefines" previously included fonts.

Example 6

```
%!PS-Adobe-3.0
%%Title: (Example of memory management)
%%DocumentNeededResources: font Helvetica Helvetica-Bold
%%+ font StoneSerif Palatino-Roman Carta Sonata
%%EndComments
%%BeginDefaults
%%PageResources: font Helvetica Helvetica-Bold StoneSerif
%%EndDefaults
%%BeginProlog
...Document prolog...
%%EndProlog
%%BeginSetup
% Include the common fonts found in most implementations
%%IncludeResource: font Helvetica
%%IncludeResource: font Helvetica-Bold
...Rest of the set up...
%%EndSetup
%%Page: 1 1
%%PageResources: font Helvetica Helvetica-Bold
%%+ font StoneSerif Palatino-Roman Carta Sonata
%%BeginPageSetup
/pagelevel save def
%%EndPageSetup
...Text that uses common fonts like Helvetica...
/fontlevel save def
%%IncludeResource: font StoneSerif
...Text that uses the StoneSerif font and/or common fonts...
%%IncludeResource: font Palatino-Roman
...Text that uses Palatino-Roman, StoneSerif and/or common fonts...
%%IncludeResource: font Carta
...Text that uses the Carta, Palatino-Roman, StoneSerif, and/or common fonts...
fontlevel restore % Ran out of room for new fonts
/fontlevel save def
%%IncludeResource: font StoneSerif
%%IncludeResource: font Palatino-Roman
%%IncludeResource: font Sonata
...Text that uses the Sonata, Palatino-Roman, StoneSerif, and/or common fonts...
fontlevel restore % Need to switch fonts
/fontlevel save def
%%IncludeResource: font StoneSerif
%%IncludeResource: font Carta
...Text that uses the Carta, StoneSerif, and/or common fonts...
pagelevel restore
showpage
%%Page: 2 2
%%PageResources: font StoneSerif Palatino-Roman
...Rest of the document...
%%EOF
```

At print time, the document manager decides there is enough memory available in the VM of the target device to hold four fonts at any one point in time and decides to optimize the document. The Helvetica and Helvetica-Bold inclusions are ignored because these fonts are available on the printer. The page level comment %%PageResources: font StoneSerif is recognized in the defaults section, indicating that the font StoneSerif is likely to be used on every page. The document manager moves the inclusion of this font to the end of the document setup and ignores all subsequent inclusion requests for StoneSerif.

The document manager also realizes that the Palatino-Roman font is only used on pages 1 and 2. This font is downloaded at the end of the page setup for each page. The Carta and Sonata fonts are used on page 1 only. However, the Carta font is downloaded twice due to the three-font memory management scheme used by the application. The document manager also moves the downloading of the Carta font to the end of the page setup. The Sonata font is used only once and is downloaded at the %%IncludeResource: font comment. Example 7 shows the resulting file:

Example 7

```
%!PS-Adobe-3.0
%%Title: (Optimized file)
%%DocumentNeededResources: font Helvetica Helvetica-Bold
%%DocumentSuppliedResources: font StoneSerif Palatino-Roman Carta Sonata
%%EndComments
%%BeginDefaults
%%PageResources: font Helvetica Helvetica-Bold StoneSerif
%%EndDefaults
%%BeginProlog
...Document prolog...
%%EndProlog
%%BeginSetup
% Include the common fonts found in most implementations
%%IncludeResource: font Helvetica
%%IncludeResource: font Helvetica-Bold
%%BeginResource: font StoneSerif
...StoneSerif font is downloaded here...
%%EndResource
...Rest of the set up...
%%EndSetup
%%Page: 1 1
%%PageResources: font Helvetica Helvetica-Bold
%%+ font StoneSerif Palatino-Roman Carta Sonata
%%BeginPageSetup
/pagelevel save def
%%BeginResource: font Palatino-Roman
...Palatino-Roman font is downloaded here...
%%EndResource
%%BeginResource: font Carta
```

```
...Carta font is downloaded here...
%%EndResource
%%EndPageSetup
...Text that uses common fonts like Helvetica...
/fontlevel save def
...Text that uses the StoneSerif font and/or common fonts...
...Text that uses Palatino-Roman, StoneSerif and/or common fonts...
...Text that uses the Carta, Palatino-Roman, StoneSerif, and/or common fonts...
fontlevel restore% Ran out of room for new fonts
/fontlevel save def
%%BeginResource: font Sonata
...Sonata font is downloaded here...
%%EndResource
...Text that uses the Sonata, Palatino-Roman, StoneSerif, and/or common fonts...
fontlevel restore % Need to switch fonts again
/fontlevel save def
...Text that uses the Carta, StoneSerif, and/or common fonts...
pagelevel restore
showpage
%%Page: 2 2
%%PageResources: font StoneSerif Palatino-Roman
%%BeginPageSetup
/pagelevel save def
%%BeginResource: font Palatino-Roman
...Palatino-Roman font is downloaded again here...
%%EndResource
...Rest of the document...
%%EOF
```

Procset note—The %%IncludeResource: procset comment must appear in the document prolog only. Procsets do not generally have to worry about **save**/**restore** pairs as in the above example. In the case of procsets, the document manager may replace the desired procset with an upwardly compatible version of the desired procset (a newer version). See section 4.6, "Comment Syntax Reference," for more details on compatible procsets. In addition, the document manager may optimize procset inclusion by replacing a procset that occurs multiple times with a single copy at the top level of a document. Example 8 shows the use of the %%IncludeResource: procset comment:

Example 8

```
%!PS-Adobe-3.0
%%Creator: Adobe Illustrator 88(TM) 1.9.3
%%For: (Joe Smith) (Adobe Systems Incorporated)
%%Title: (Example.art)
%%CreationDate: (2/08/90) (8:30 am)
%%DocumentNeededResources: procset Adobe_packedarray 0 0
%%+ procset Adobe_cmykcolor 0 0 Adobe_cshow 0 0 Adobe_customcolor 0 0
%%+ procset Adobe_Illustrator881 0 0
%%+ font StoneSerif
%%EndComments
%%BeginProlog
%%IncludeResource: procset Adobe_packedarray 0 0
%%IncludeResource: procset Adobe_cmykcolor 0 0
%%IncludeResource: procset Adobe_cshow 0 0
%%IncludeResource: procset Adobe_customcolor 0 0
%%IncludeResource: procset Adobe_Illustrator881 0 0
%%EndProlog
...Rest of the document...
%%EOF
```

6.3 Requirement Page Comments

Some of the following comments that request particular page media, requirements, or resources may appear in the defaults section or in a particular page. If these comments fall within the defaults section of the document file (%%BeginDefaults to %%EndDefaults), they may be construed to be in effect for the entire print job. If they are found within the page-level comments for a page, they should only be in effect for that page. See %%Begin(End)Defaults for more details on page defaults.

%%PageFonts: { <fontname> ... } | (atend)

Indicates the names of all fonts used on the current page. The notation (atend) is permissible. In that case, the list of fonts must be provided after the %%PageTrailer comment. Also see the %%DocumentFonts: comment.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%PageResources: *comment instead.*

%%PageFiles: { <filename> ... } | (atend)

Indicates the names of all files used on the current page. This should be used only if file inclusion is required of the document manager—that is, if there are subsequent instances of the %%IncludeFile: comment on that particular page. See also %%DocumentNeededFiles: and %%DocumentSuppliedFiles: comments.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%PageResources: *comment instead.*

%%PageMedia: <medianame>
<*medianame*> ::= <*text*> (Name of desired paper media)

Indicates that the paper attributes denoted by *medianame* are invoked on this page. The *medianame* is specified by the %%DocumentMedia: comment at the beginning of the document. This comment can pertain to either a page or a document depending on the position of the comment (for example, either in the page itself or in the defaults section). See also the %%DocumentMedia: and %%Begin(End)Defaults comments.

In Example 9, a one-hundred page report is printed on regular white and heavy yellow paper. Ninety-nine of the pages use the white paper so the %%PageMedia: comment is found in the defaults section, denoting that the default media for this document is white paper. The white paper is set using the **setpagedevice** operator in the document setup. The cover page is the only page to use the yellow paper, and states so via the %%PageMedia:

comment that appears after the first %%Page: comment. Note the use of the **currentpagedevice** operator to facilitate the restoration of the white-paper device after the cover page.

Example 9

```
%!PS-Adobe-3.0
%%Title: (Example of %%PageMedia: as a page default)
%%DocumentMedia: Regular 612 792 75 white ( )
%%+ Cover 612 792 244 yellow DeptLetterHead
%%Pages: 100
%%LanguageLevel: 2
%%EndComments
%%BeginDefaults
%%PageMedia: Regular
%%EndDefaults
%%BeginProlog
...Prolog definitions...
%%EndProlog
%%BeginSetup
<<                                          % Attribute tray numbers to
  /InputAttributes <<                       % the particular media
    0 << /PageSize [612 792] /MediaWeight 75 /MediaColor (white) >>
    1 << /PageSize [612 792] /MediaWeight 244
         /MediaColor (yellow) /MediaType (DeptLetterHead) >>
  >>
>> setpagedevice

<< /MediaColor (white) >> setpagedevice     % Set the white paper to be the
%%EndSetup                                  % default for the document
%%Page: Cover 1
%%PageMedia: Cover
%%BeginPageSetup
/olddevice currentpagedevice def
<< /MediaColor (yellow) >> setpagedevice    % Set up the yellow paper
/pagelevel save def                         % for this page
%%EndPageSetup
...Mark the cover page...
pagelevel restore
showpage
%%PageTrailer
olddevice setpagedevice                     % Restore the white paper
%%Page: 1 2
...Rest of the document...                  % No %%PageMedia:
%%EOF                                       % comment, white letter paper
                                            % is the default
```

%%PageRequirements: **<requirement> [(<style>)] ...**

<*requirement*> ::= collate | color | duplex | faceup | fax | fold | jog | manualfeed | numcopies | punch | resolution | rollfed | staple

<*style*> ::= <*text*>

This is the page-level invocation of a combination of the options listed in the %%Requirements: comment. It takes precedence over any document requirements set during the document setup. This comment can pertain to a page or a document depending on the position of the comment (either in the page itself or in the defaults section). See the %%Requirements: and %%Begin(End)Defaults comments.

%%PageResources: **{ <resource> ... } | (atend)**

This comment indicates the names and values of all resources that are needed or supplied on the present page (procsets are an exception; they need not be listed). This comment can pertain to an individual page or a document, depending on the location of the comment. For example, the comment may be in the page itself or in the document defaults section. See the %%DocumentSuppliedResources:, %%DocumentNeededResources:, and %%Begin(End)Defaults comments.

7 Color Separation Conventions

Level 2 implementations and Level 1 implementations that contain the CMYK color extensions to the PostScript language provide more complete color functionality than the RGB color model in Level 1. There are corresponding color separation comments that programs producing PostScript language documents with color operators should use. Color separation applications can use these comments as an aid in proper color determination and to identify process color specific portions of PostScript language code. These comments can also be used to enable applications to communicate spot color usage.

Note *These comments do not address the use of CIE based and special color spaces. Expect future versions of the DSC to do so.*

7.1 Color Header Comments

%%CMYKCustomColor: <cya> <mag> <yel> <blk> <colorname>

<*cya*> :: = <*real*>	(Cyan percentage)
<*mag*> ::= <*real*>	(Magenta percentage)
<*yel*> ::= <*real*>	(Yellow percentage)
<*blk*> ::= <*real*>	(Black percentage)
<*colorname*> ::= <*text*>	(Custom color name)

This comment provides an *approximation* of the custom color specified by *colorname*. The four components of cyan, magenta, yellow, and black must be specified as numbers from 0 to 1 representing the percentage of that process color. The numbers are similar to the arguments to the **setcmykcolor** operator. The *colorname* follows the same custom color naming conventions as the %%DocumentCustomColors: comment.

%%DocumentCustomColors: { <colorname> ... } | (atend)

<*colorname*> ::= <*text*>	(Custom color name)

This comment indicates the use of custom colors in a document. An application arbitrarily names these colors, and their CMYK or RGB approximations are provided through the %%CMYKCustomColor: or %%RGBCustomColor: comments in the body of the document. Normally, the *colorname* specified can be any arbitrary string except Cyan, Magenta, Yellow, or Black. If imaging to a specific process layer is desired, these names may be used.

%%DocumentProcessColors: { <color> ... } | (atend)

<*color*> ::= Cyan | Magenta | Yellow | Black

This comment marks the use of process colors in the document. Process colors are defined to be Cyan, Magenta, Yellow, and Black. This comment is used primarily when producing color separations. See also %%PageProcessColors:.

%%RGBCustomColor: <red> <green> <blue> <colorname>

<*red*> ::= <*real*>	(Red percentage)
<*green*> ::= <*real*>	(Green percentage)
<*blue*> ::= <*real*>	(Blue percentage)
<*colorname*> ::= <*text*>	(Custom color name)

This comment provides an *approximation* of the custom color specified by *colorname*. The three components of red, green, and blue must be specified as numbers from 0 to 1 representing the percentage of that process color. The numbers are similar to the arguments to the **setrgbcolor** operator. The *colorname* follows the same custom color naming conventions as the %%DocumentCustomColors: comment.

7.2 Color Body Comments

%%BeginCustomColor: <colorname>

<*colorname*> ::= <*text*>	(Custom color name)

%%EndCustomColor (no keywords)

These comments specify that the PostScript language code fragment enclosed within should be interpreted only when rendering the separation identified by *colorname*. The *colorname* here is any text string except Cyan, Magenta, Yellow, and Black (see the exception in %%DocumentCustomColors:).
During color separation, the code between these comments must only be downloaded during the appropriate pass for that custom color. Intelligent printing managers can save considerable time by omitting code within these bracketing comments during any other separations. The document composition software must be extremely careful to correctly control overprinting and knockouts if these comments are employed, because the enclosed code may or may not be executed.

Note *In the absence of a document manager that understands these comments, the document will print incorrectly. These comments should be used only if the environment supports such a document manager.*

%%BeginProcessColor: **<color>**
<*color*> ::= Cyan | Magenta | Yellow | Black

%%EndProcessColor (no keywords)

These comments specify that the PostScript language code fragment enclosed within should be interpreted only when rendering the separation identified by *color*. During color separation, the code between these comments must be downloaded only during the appropriate pass for that process color. Intelligent printing managers can save considerable time by omitting code within these bracketing comments on the other three separations. The document composition software must be extremely careful to correctly control overprinting and knockouts if these comments are employed, because the code may or may not be executed.

Note *In the absence of a document manager that understands these comments, the document will print incorrectly. These comments should only be used if the environment supports such a document manager.*

7.3 Color Page Comments

%%PageCustomColors: **{ <colorname> ... } | (atend)**
<*colorname*> ::= <*text*> (Custom color name)

This comment indicates the use of custom colors in the page. An application arbitrarily names these colors, and their CMYK or RGB approximations are provided through the %%CMYKCustomColor: or %%RGBCustomColor: comments in the body of the document. See the %%DocumentCustomColors: comment.

%%PageProcessColors: **{ <color> ... } | (atend)**
<*color*> ::= Cyan | Magenta | Yellow | Black

This comment marks the use of process colors in the page.
Process colors are defined as Cyan, Magenta, Yellow, and Black.
See the %%DocumentProcessColors: comment.

8 Query Conventions

A *query* is any PostScript language program segment that generates and returns information back to the host computer across the communications channel *before* a document can be formatted for printing. This might result from the execution of any of the **=, ==, print** or **pstack** operators, for instance. In particular, this definition covers information that is expected back from the PostScript printer for decision-making purposes. Such decision-making might include the generation of font lists or inquiries about the availability of resources, printer features, or the like.

All query conventions consist of a *begin* and *end* construct, with the keywords reflecting the type of query. For all of them, the %%?EndQuery comment should include a field for a *default* value, which document managers must return if they cannot understand or do not support query comments. The value of the default is entirely application dependent, and an application can use it to determine specific information about the spooling environment, if any, and to take appropriate default action.

8.1 Responsibilities

A document manager that expects to be able to interpret and correctly spool documents conforming to DSC version 3.0 must, at a minimum, perform certain tasks in response to these query conventions. In general, it must recognize the queries, remove them from the print stream, and send some reply back to the host. If a document manager cannot interpret the query, it must return the value provided as the argument to the %%?EndQuery comment.

A query can be recognized by the sequence %%?Begin followed by any number of characters (up to the 255 maximum per line, by convention) through the end-of-line indication (the % is decimal ASCII 37, and the ? is decimal ASCII 63). The end of the query is delimited by the sequence %%?End followed by some keywords, and optionally followed by a colon (: is decimal ASCII 58) and the default response to the query (any text through end-of-line). A document manager should try to recognize the full query keyword, such as %%?BeginResourceQuery:, if it can, but it is obligated at least to respond to any validly formed query.

If a more intelligent query handling interface is desired, the document manager must recognize which printer the application is printing to (the %%DocumentPrinterRequired: comment may be helpful in this case). By using the PPD file for that particular printer, the known printer network configuration, and the printer status, the document manager should be able to answer the query.

8.2 Query Comments

%!PS-Adobe-3.0 Query (no keywords)

A PostScript language query must be sent as a separate job to the printer to be fully spoolable. This means that an *end-of-file* indication must be sent immediately after the query job. A query job must always begin with the %!PS-Adobe-3.0 Query convention, which further qualifies the file as being a special case of a version 3.0 conforming PostScript language file. A query job contains only query comments, and need not contain any of the other standard structuring conventions. A document manager must be prepared to extract query information from any print file that begins with this comment convention. A document manager must fully parse a query job file until the EOF indication is reached.

Note *It is permissible to include more than one query in a print job, but it is not permissible to include queries within the body of a regular print job. It cannot be guaranteed that a document manager can properly handle a print job with embedded queries.*

%%?BeginFeatureQuery: **<***featuretype***> [<***option***>]**

<*featuretype*> ::= <*text*> (Requested feature)

<*option*> ::= <*text*> (Feature option)

%%?EndFeatureQuery: **<***default***>**

<*default*> ::= <*text*> (Default response)

This query provides information that describes the state of some specified, printer-specific feature as defined by the PostScript printer description (PPD) file. The *featuretype* field identifies the keyword as found in the PPD file. The standard response varies with the feature and is defined by the printer's PPD file. In general, the value of the <*featuretype*> or the value of <*option*> associated with the feature should be returned. In the example that follows, the PPD file keywords True or False are returned:

```
%%?BeginFeatureQuery: *InputSlot manualfeed
 statusdict /manualfeed known {
    statusdict /manualfeed get { (True) }{ (False) } ifelse
 }{
     (None)
} ifelse = flush
%%?EndFeatureQuery: Unknown
```

%%?BeginFileQuery: **<***filename***>**

%%?EndFileQuery: **<***default***>**
<*default*> ::= <*text*> (Default response)

The PostScript language code between these comments causes the printer to respond with information describing the availability of the specified file. This presumes the existence of a file system that is available to the PostScript interpreter, which is not the case on all implementations. The standard response consists of a line containing the file name, a colon, and either Yes or No, indicating whether the file is present.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%?Begin(End)ResourceQuery: *comments instead.*

%%?BeginFontListQuery (no keywords)

%%?EndFontListQuery: **<***default***>**
<*default*> ::= <*text*> (Default response)

Provides a PostScript language sequence to return a list of all available fonts. It should consult the **FontDirectory** dictionary and any mass storage devices available to the interpreter. The list need not be in any particular order, but each name should be returned separated by a slash / character. This is normally the way the PostScript **==** operator returns a font name. All white space characters should be ignored. The end of the font list must be indicated by a trailing * (asterisk) sign on a line by itself (decimal ASCII 42).

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%?Begin(End)ResourceListQuery: *comments instead.*

%%?BeginFontQuery: **<***fontname***> ...**

%%?EndFontQuery: **<***default***>**
<*default*> ::= <*text*> (Default response)

This comment provides a PostScript language query that should be combined with a particular list of font names being sought. It looks for any number of names on the stack and prints a list of values depending on whether the font is known to the PostScript interpreter. The font names must be provided on the operand stack by the document manager. This is done by simply sending the names, with leading slash / characters, before sending the query itself.

To prevent the document manager from having to keep track of the precise order in which the values are returned and to guard against errors from dropped information, the syntax of the returned value */FontName*:Yes or /*FontName*:No, with no space between the colon and the following word.

Each font in the list is returned this way. The slashes delimit the individually returned font names, although newlines should be expected (and ignored) between them. A final * (asterisk) character follows the returned values.

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%?Begin(End)ResourceQuery: *comments instead.*

%%?BeginPrinterQuery (no keywords)

%%?EndPrinterQuery: **<***default***>**
<*default*> ::= <*text*> (Default response)

This comment delimits PostScript language code that returns information describing the printer's *product name*, *version*, and *revision* numbers. The standard response consists of the printer's product name, version, and revision strings, each of which must be followed by a newline character, which must match the information in the printer's printer description file. This comment may also be used to identify the presence of a spooler, if necessary. In the following example the default response as represented in the %%?EndPrinterQuery: line is the word spooler, which would be returned by spooling software that *did not* have a specific printer type attached to it.

```
%%?BeginPrinterQuery
statusdict begin
    revision == version == productname == flush
end
%%?EndPrinterQuery: spooler
```

%%?BeginProcSetQuery: **<***procname***>**

%%?EndProcSetQuery: **<***default***>**
<*default*> ::= <*text*> (Default response)

These comments delimit a procset query. The combination of the *name, version,* and *revision* fields must uniquely identify the procset. The standard response to this query consists of a line containing any of the values 0, 1, 2 where a value of 0 means the procset is *missing*, a value of 1 means the procset is *present and OK*, and a value of 2 indicates the procset is present but is an incompatible version. Note that methods for procset queries are procset specific.

```
%%?BeginProcSetQuery: adobe_distill 1.1 1
/adobe_distill_dict where {
  begin mark VERSION (1.) anchorsearch {(1)}{(2)} ifelse cleartomark
  end
}{
  (0)
} ifelse print flush
%%?EndProcSetQuery: unknown
```

Note *This comment is provided for backward compatibility and may be discontinued in later versions of the DSC. Use the more general* %%?Begin(End)ResourceQuery: *comments instead.*

%%?BeginQuery: **<***identifier***>**
<*identifier*> ::= <*text*> (Query identifier)

%%?EndQuery: **<***default***>**
<*default*> ::= <*text*> (Default response)

These comments are for very general purposes and may serve any function that the rest of the query conventions, which are very specific, do not adequately cover. To understand and intelligently respond to a query, a document manager must semantically understand the query. Therefore, specific keywords, such as %%?BeginPrinterQuery, are used. When the generic %%?BeginQuery comment is encountered, a spooler may be forced to return the default value. The comment is included primarily for large installations that must implement specific additional queries not covered here, and which will likely implement the document composition software and the document manager software.

%%?BeginResourceListQuery: font | file | procset | pattern | form | encoding

%%EndResourceListQuery: **<***text***>**

These comments delimit a segment of PostScript language code that returns a list of all available resources. The arguments specify which type of resources to return. The code that these comments delimit should consult local VM,

global VM, and any mass storage devices available to compile a complete list of resources. The resulting list need not be in any particular order, but the syntax of the returned values is the *resource type* followed by the *resource name*. The end of the resource list must be indicated by a trailing * (asterisk) on a line by itself.

Note that font names must be returned with a slash / character in front of each font name.

Note *The use of this type of query is discouraged because it can be time consuming for interpreters with many accessible resources (for example, a printer with a hard disk attached). It is far better to query for individual resources by using the* %%?Begin(End)ResourceQuery: *comment.*

%%?BeginResourceQuery: <resource>...

%%?EndResourceQuery: <default>
<*default*> ::= <*text*> (Default response)

The PostScript language code between these comments causes the printer to respond with information describing the availability of the specified resources. This code looks for any number of resource names on the stack, and prints a list of values depending on whether the resource is known to the PostScript interpreter.

The document manager could also process this query by using information known about the print network and current printer status. To reduce the overhead involved in keeping track of the precise order in which values are returned, and to guard against errors from dropped information, the syntax of the returned value is the *resource type* and *name* followed by a colon, a space and then a yes or a no. The end of the list should be denoted by a *.

Note *It is recommended that a separate resource query be used for each type of resource.*

A file resource query presumes that a file system is available to the PostScript interpreter. This is not the case in all implementations. Example 10 shows a typical font resource query:

Example 10

```
%!PS-Adobe-3.0 Query
%%Title: (Resource query for specified fonts)
%%?BeginResourceQuery: font Times-Roman Adobe-Garamond StoneSerif
/Times-Roman
/Adobe-Garamond
/StoneSerif
%%BeginFeature: *?FontQuery
save 4 dict begin /sv exch def
/str (fonts/                ) def
/st2 128 string def
{
  count 0 gt {
    dup st2 cvs (Font /) print print
    dup FontDirectory exch known
    { pop (: Yes) }
    { str exch st2 cvs
      dup length /len exch def
      6 exch putinterval str 0 len 6 add getinterval mark exch
      { } st2 filenameforall counttomark
      0 gt {? cleartomark (: Yes) }{ cleartomark (: No) }ifelse
    } ifelse = flush
  }{ exit } ifelse
} bind loop
(*) = flush
sv end restore
%%EndFeature
%%?EndResourceQuery:  Unknown
%%EOF
```

The output from this sample program could be:

```
Font /StoneSerif: Yes
Font /Adobe-Garamond: No
Font /Times-Roman: No
*
```

%%?BeginVMStatus (no keywords)

%%?EndVMStatus: ***<default>***
<*default*> ::= <*text*> (Default response)

This comment delimits PostScript language instructions that return the state of the PostScript interpreter's VM. The standard response consists of a line containing the results of the PostScript language **vmstatus** operator as shown in Example 11:

Example 11

```
%!PS-Adobe-3.0 Query
%%Title: (VM status query)
%%?BeginVMStatus
vmstatus
(Maximum: ) print =
(Used: ) print =
(Save Level: ) print = flush
%%?EndVMStatus: Unknown
%%EOF
```

9 Open Structuring Conventions

There is an open extension mechanism for the DSC comments. Its purpose is to enable other vendors to extend the functionality of the DSC without having to rely on Adobe to amend the official specification.

Vendors may need or want to embed extra information in a file beyond the comments that Adobe has already specified. To facilitate this and to minimize conflicts and difficulties for the vendor, Adobe maintains a registry of comment prefixes that are allocated to vendors, and these comments may be used in any way that is meaningful to those vendors. You may contact the registry at the following address:

DSC Coordinator
Developer Support Dept.
Adobe Systems Incorporated
345 Park Avenue
San Jose, CA 95110

9.1 The Extension Mechanism

All existing Adobe-specified comments in the DSC begin with the same prefix, except one. Here is a quick summary of the syntax of existing comments:

The first line of a PostScript language file must, by convention, begin with the characters %! (percent and exclamation, often referred to as “percent-bang”). If the file is a conforming file, meaning that it conforms to the DSC version 3.0, then it is further qualified with PS-Adobe-3.0. This may be optionally continued by some special keywords, such as EPSF or ExitServer, to identify the entire file as a special instance. The first line of a PostScript language file may look something like this:

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

This is the only Adobe-defined comment that does *not* begin with two percent signs.

All remaining structuring conventions, in their various forms, are represented as comments beginning with *two* percent signs (%%) as the first characters on the line.

The extension mechanism for the open structuring conventions is to use one percent character followed immediately by a *vendor-specific* prefix of up to five characters. Beyond those five characters the vendor who has registered the prefix is responsible for the comments. The comment is terminated at the end of the line.

Open structuring conventions may be used much like the existing DSC and have similar syntax and philosophy. Here are some examples of *fictitious* comments from made-up company prefixes:

```
%GCRImageName: myimage.ps
%BCASpoolerName: local_spool 1.0
%BCACoverStock: 10129
%BCADocumentOrigin: (New York Office)
```

Restrictions

Adobe does not specify where in the document open structuring convention comments can appear. However, the comments must not conflict in any way with the regular parsing of document structuring conventions, and their specification and use is otherwise truly open.

If these vendor-specific comments interact in some meaningful way with the DSC, this interaction should be clearly specified by the creator of the comments, and the description should specify the version number of the DSC with which they interact.

The new comments, however implemented, should still follow the conforming files restrictions discussed in section 3, “DSC Conformance.”

Parsing Rules

Although the exact syntax of the vendor-specific comments is up to the vendor, we strongly recommend adhering to the existing conventions and parsing rules to simplify the task of writing parsing software.

Note *The syntax and parsing rules for vendor-specific comments are up to the vendor, and you should contact the vendor for details. The rules and details supplied in this document are guidelines and suggestions that are recommended, but are not enforced by Adobe.*

10 Special Structuring Conventions

There are two comments that do not readily fall into the other comment categories. They are listed below, along with a description of when they should be used.

%%BeginExitServer: ***<password>***
<*password*> ::= <*text*>

%%EndExitServer (no keywords)

These comments delimit the PostScript language sequence that causes the rest of the file to be executed as an unencapsulated job (see section 3.7.7, "Job Execution Environment" of the *PostScript Language Reference Manual, Second Edition*). This convention is used to flag any code that sets up or executes the **exitserver** or **startjob** operators, so a document manager can recognize and remove this sequence if necessary. The %%Begin(End)ExitServer comments may be used with the %%EOF requirement convention to pinpoint where the document manager should send an end-of-file indication. See the %!PS-Adobe-3.0 comment. PostScript language jobs that use **exitserver** or **startjob** should be specially flagged with the %!PS-Adobe-3.0 ExitServer notation. An example of appropriate use is shown in the following example:

```
%!PS-Adobe-3.0 ExitServer
%%Title: (Example of exitserver usage)
%%EndComments
%%BeginExitServer: 000000
serverdict begin 000000 exitserver
%%EndExitServer
...PostScript language instructions to perform
persistent changes...
%%EOF
```

Appendix A: Changes Since Earlier Versions

This content of this document is the same as the specification in Appendix G of the *PostScript Language Reference Manual, Second Edition*. This document tracks the changes made in subsequent printings of the *PostScript Language Reference Manual, Second Edition* and those that are listed in Technical Note #5085, "Updates to the PostScript Language Reference Manual, Second Edition."

A.1 Changes Since Earlier Versions

The following section details changes made to the DSC specification since version 1.0 (Appendix C in the first edition of the *PostScript Language Manual*). These changes are important to document managers that may wish to allow backward compatibility with previous versions of this specification.

A.1.1 Changes Since Version 1.0

In DSC version 1.0, there were several comment conventions that were required to minimally conform to that version of the specification. These comments were:

```
%%DocumentFonts:
%%EndProlog
%%Page:
%%Trailer
```

As of version 2.1, there no longer are any *required* comments. All comments are optional in the sense that they may not be appropriate in a given situation. The only rule is to make sure to use them correctly.

The following comments were added as of version 2.1:

```
%%Begin(End)Binary:
%%Begin(End)CustomColor:
%%Begin(End)Document:
%%Begin(End)ExitServer:
%%Begin(End)Feature:
```

%%Begin(End)File:
%%Begin(End)Font:
%%Begin(End)Object:
%%Begin(End)PageSetup:
%%Begin(End)PaperSize:
%%Begin(End)ProcessColor:
%%Begin(End)ProcSet
%%Begin(End)Setup
%%CMYKCustomColor:
%%DocumentCustomColors:
%%DocumentNeededFiles:
%%DocumentNeededFonts:
%%DocumentNeededProcSets:
%%DocumentPaperColors:
%%DocumentPaperSizes:
%%DocumentPaperForms:
%%DocumentPaperWeights:
%%DocumentPrinterRequired:
%%DocumentProcSets:
%%DocumentProcessColors:
%%DocumentSuppliedFiles:
%%DocumentSuppliedFonts:
%%DocumentSuppliedProcSets:
%%ExecuteFile:
%%IncludeFile:
%%IncludeFont:
%%IncludeProcSet:
%%EOF
%%Feature:
%%PageBoundingBox:
%%PageCustomColors:
%%PageFonts:
%%PageFiles:
%%PageProcessColors:
%%PageTrailer
%%PaperColor:
%%PaperForm:
%%PaperSize:
%%PaperWeight:
%%ProofMode:
%%Requirements:
%%RGBCustomColor:
%%Routing:
%%?Begin(End)FeatureQuery:
%%?Begin(End)FileQuery:
%%?Begin(End)FontQuery:
%%?Begin(End)FontListQuery:
%%?Begin(End)ProcSetQuery:
%%?Begin(End)PrinterQuery:
%%?Begin(End)Query:
%%?Begin(End)VMStatus:

The following comment was discontinued in version 2.1 and should be ignored by document managers:

%%ChangeFont:

A.1.2 Changes Since Version 2.1

The DSC version 3.0 specification has been reorganized as a whole to better present the concepts. The first half of the specification is a how-to guide and discusses why the comments should be used. The second half is a reference, detailing the comments.

The introduction introduces the concepts of a document manager and how a document manager might use the comments.

A new section talks about the various services a document can receive from a document manager. These services can be obtained through proper use of the DSC comments. Services include spooling, banner and trailer pages, print logging, resource inclusion, resource downloading, resource optimization, error reporting and recovery, printer rerouting, feature inclusion, parallel printing, color breakout, page reversal, n-up printing, range printing, collated printing, and overlays. See section 2, “Document Manager Services.”

The section detailing DSC conformance has been expanded and is more precise. A document either conforms or does not conform to this specification. See section 3, “DSC Conformance.”

A new section describing proper document structure was added. In particular, the placement of various comments in the document is discussed as are restrictions on the prolog and script. See section 4, “Document Structure Rules.”

A section detailing the breakdown of conventions into different categories was added, as well as detailed explanations of header, body and page comment types. The comments are arranged in the reference section of the document according to these categories. See section 4.5, “Convention Categories.”

The syntax of the DSC comments was qualified in Backus-Naur form (BNF) to avoid ambiguities. A new section of the document talks about BNF and defines some elementary types. See section 4.6, “Comment Syntax Reference.”

The open structuring conventions are new as of this version. They define an extensible mechanism for defining vendor-specific comments. See section 9, “Open Structuring Conventions.”

New Comments For Version 3.0

The following comments were added as of version 3.0:

%%Begin(End)Data:
%%Begin(End)Defaults
%%Begin(End)Emulation:
%%Begin(End)Preview:
%%BeginProlog
%%Begin(End)Resource:
%%Copyright:
%%DocumentData:
%%DocumentMedia:
%%DocumentNeededResources:
%%DocumentSuppliedResources:
%%Emulation:
%%Extensions:
%%IncludeDocument:
%%IncludeFeature:
%%IncludeResource:
%%LanguageLevel:
%%OperatorIntervention:
%%OperatorMessage:
%%Orientation:
%%PageMedia:
%%PageOrder:
%%PageOrientation:
%%PageRequirements:
%%PageResources:
%%Version
%%VMlocation:
%%VMusage:
%%?Begin(End)ResourceQuery:
%%?Begin(End)ResourceListQuery:

There are three justifications for the addition of the %%BeginProlog comment. Previously, the beginning of the prolog section of the document was implicitly declared after the %%EndComments comment. This was confusing in the case of EPSI files that needed to insert the EPSI preview after the comments and before the prolog, which was defined as the first %%BeginProcSet: comment. In addition, there may be instances when a document does not need formal procset definitions, but needs a prolog. Finally, in the interest of language purity, a corresponding %%Begin comment is necessary for each %%End comment. Expect to see this pairing of comments in future revisions of the DSC.

Changes to Existing Comments

%!PS-Adobe-3.0
In addition to changing the version number from 2.1 to 3.0, the new EPSF version number was added, as well as a general format keyword for resources.

%%Pages:
The optional *pageorder* number at the end of the comment is no longer recommended (-1 indicated descending order, 0 indicated special order, and 1 indicated ascending order). There have been cases of conflicts between pre-knowledge of page orders and page numbers; in other words, an application may not know the number of pages, and wishes to defer this comment to the end of the document, but it may already know the page order. Previewers and other document managers gain an advantage if they know the page order as soon as possible. If page order must be specified, it is recommended that it be done using the %%PageOrder: comment.

%%Begin(End)Binary:
There has been some confusion with this comment. Both hex and 8-bit binary data has been seen between these comments. There also have been some cases in which the byte count argument to this comment has been used to specify the number of lines of data. A new comment, %%Begin(End)Data:, has been introduced to deal with these ambiguities. The new comment may also be extended in future versions of the DSC to deal with compression and other filters, so a document manager can handle special filtering on Level 1 implementations.

%%Requirements:
The idea of option *styles* is introduced. These styles modify the requirement option in some manner. For example, punch(3) indicates that the printer needs to support 3 hole punching. Similarly, duplex(tumble) indicates that the printer must be able to perform tumble duplexing.

New options include manualfeed, numcopies, collate, jog, faceup, resolution, rollfed, fax, and punch. They reflect the additional functionality added by the Level 2 **setpagedevice** operator.

Deleted options include simplex, punch3, punch5. The simplex option is redundant because if duplex is not specified as a requirement, simplex is implied. The punch3 and punch5 options have been superceded by the idea of style modifiers (see above).

%%Begin(End)Document:
There has been a note added to this comment indicating that feature and resource requirements of an included document should be inherited by the including document.

%%ExecuteFile:
This comment has been renamed %%IncludeDocument to better reflect its meaning.

%%Feature:
This comment has been renamed %%IncludeFeature: to more clearly express its dependence on the document manager.

Discontinued Comments For Version 3.0

%%BeginPaperSize:
%%EndPaperSize
The comments %%BeginFeature: and %%EndFeature should be substituted.

%%DocumentPaperColors:
%%DocumentPaperForms:
%%DocumentPaperSizes:
%%DocumentPaperWeights:
These comments have been replaced by the single %%DocumentMedia: comment. This new comment addresses two shortcomings of DSC version 2.1. First, the new comment provides the linkage among the various parameters describing an output medium. Second, a generalized portable methodology for describing paper is provided.

For document managers concerned with backward compatibility, the following comments

```
%%DocumentPaperColors: white buff pink
%%DocumentPaperForms: Plain Plain CorpLetterHead
%%DocumentPaperSizes: letter letter legal
%%DocumentPaperWeights: 20 65 20
```

can be converted to

```
%%DocumentMedia: Wplain 612 792 75 white
%%+ Bplain 612 792 244 buff
%%+ CLHpink 612 1008 75 pink CorpLetterHead
```

Note that in version 2.1 there was no explicit link among the listed arguments and the other comments. The document manager will have to use a best-guess method of conversion or ignore these comments entirely.

%%PaperColor:
%%PaperForm:
%%PaperSize:
%%PaperWeight:
The individual paper-request comments are now replaced with the single %%PageMedia: comment.

Document managers trying to maintain backward compatibility should match the %%DocumentMedia: comment with its old counterparts (see above). %%PageMedia: will use the names of the different media specified in %%DocumentMedia: to specify changes in media. The paper comments for forms, colors, and weights should be replaced with the corresponding %%PageMedia: comment.

Appendix B: DSC Version 3.0 Summary

B.2 DSC Version 3.0 Summary

The following summary lists the comments that comprise version 3.0 of the document structuring conventions.

Note *Some comments in this document may be discontinued in future versions of the DSC and are not found in this list. However, they are in the body of the document for backward compatibility with existing applications and document managers. Their use is discouraged; they will eventually be omitted from the specification.*

B.2.1 General Conventions

General Header Comments

%!PS-Adobe-3.0
%%BoundingBox:
%%Creator:
%%CreationDate:
%%DocumentData:
%%DocumentPrinterRequired:
%%Emulation:
%%EndComments
%%Extensions:
%%For:
%%Version:
%%Copyright:
%%LanguageLevel:
%%OperatorIntervention:
%%OperatorMessage:
%%Orientation:
%%Pages:
%%Routing:
%%Title:

General Body Comments

%%+
%%Begin(End)Data:
%%Begin(End)Defaults
%%Begin(End)Emulation:
%%Begin(End)ExitServer:
%%Begin(End)Preview:
%%Begin(End)Prolog
%%Begin(End)Setup

General Page Comments

%%Begin(End)Object:
%%Begin(End)PageSetup:
%%Page:
%%PageBoundingBox:
%%PageOrientation:

General Trailer Comments

%%PageTrailer
%%Trailer
%%EOF

B.2.2 Requirement Conventions

Requirement Header Comments

%%DocumentMedia:
%%DocumentNeededResources:
%%DocumentSuppliedResources:
%%Requirements:
%%ProofMode:
%%VMlocation:
%%VMusage:

Requirement Body Comments

%%Begin(End)Document:
%%Begin(End)Feature:
%%Begin(End)Resource:
%%EOF
%%IncludeDocument:
%%IncludeFeature:
%%IncludeResource:

Requirement Page Comments

%%PageMedia:
%%PageRequirements:
%%PageResources:

B.2.3 Color Separation Conventions

Color Header Comments

%%CMYKCustomColor:
%%DocumentCustomColors:
%%DocumentProcessColors:
%%RGBCustomColor:

Color Body Comments

%%Begin(End)CustomColor:
%%Begin(End)ProcessColor:

Color Page Comments

%%PageCustomColors
%%PageProcessColors

B.2.4 Query Conventions

%!PS-Adobe-3.0 Query
%%?Begin(End)FeatureQuery:
%%?Begin(End)PrinterQuery:
%%?Begin(End)Query:
%%?Begin(End)ResourceQuery:
%%?Begin(End)ResourceListQuery:
%%?Begin(End)VMStatus:

Index

Symbols

A

B

C

D

E

F

G

H

I

L

N

O

P

Q

R

S

T

U

V