DOCUMENT STRUCTURING CONVENTIONS Specification Version 2.1

January 16, 1989
PostScript® Developer Tools & Strategies Group

Adobe Systems Incorporated
1585 Charleston Road PO Box 7900
Mountain View, CA 94039-7900
(415) 961-4400

PN LPS5001

POSTSCRIPT®

DOCUMENT STRUCTURING CONVENTIONS
Specification
Version 2.1

January 16, 1989
PostScript® Developer Tools and Strategies Group

1. INTRODUCTION

This document provides an extension to the published PostScript language document structuring conventions (Version 1.0). These extended comment conventions will provide print spoolers, servers and post-processors (known collectively in this document as *document managers*) with additional information about the structural organization and resource requirements of a PostScript language document file.

The Document Structuring Conventions are provided to address issues of PostScript language resource management at a site-wide level, and to allow software systems to maintain print files independent of printer-specific issues such as available paper sizes, fonts, output bins or file systems. The structuring conventions are also designed to work with the Adobe PostScript Printer Description files (PPD files), which provide the PostScript language extensions for specific printer features in a regular, parsable format. For example, these files will include information about how to invoke manual feed or different paper sizes, and they will also include information about the fonts that are built into the ROM of each printer. The structuring conventions will work in tandem with these files to provide a method of specifying and invoking printer features.

The Adobe Document Structuring Conventions allow complete cooperation between document handling mechanisms at all levels. They provide a layer of structural information which is, in a sense, superimposed on a PostScript language document file. This level of structure is used by document handling software without the need for parsing the PostScript language directly. This layer is *cooperative* rather than *enforced*, in that the comments themselves are always ignored by the PostScript interpreter, and they may or may not be interpreted by other software. A document is said to be *conforming* if it observes the Adobe® Document Structuring Conventions, and the conforming document can be expected to adhere to specific structuring constraints in that event.

It is assumed that document managers may be capable of producing prologue, font and disk file resources, although that is not required to make effective use of the structuring conventions. It is also assumed that the reader is mostly familiar with the motivation, goals and design of the existing structuring conventions. For further information, consult *Appendix C* in the *PostScript Language Reference Manual*. All of the existing comment conventions have been included in this document for reference and continued use.

2. HISTORY AND MOTIVATION

One of the most important levels of document structuring in the PostScript language is the distinction between the *Document Prologue* and the *Document Script*. The prologue is typically a set of procedure definitions appropriate for the set of operations required by a document composition system, and the script is the software-generated data that represents a particular document. A *conforming* PostScript language document description will usually have a clearly defined prologue and script (see the **%%EndProlog** structure convention) which should be distinct within a given print file. In other words, there should be nothing "executed" in the prologue, and "no definitions" in the script. Furthermore, a document *script* may be divided again into several *pages*, each of which will be *functionally independent* of the other pages. This means that the individual pages should be able to execute in any order and may be physically rearranged without affecting the resulting document. Pages may be printed in parallel as long as the prologue definitions are made available to each page. Note further that one document description may *wholly contain* another document description as part of its script without disrupting the semantics of the structuring. For instance, a PostScript language document may be placed into another document as an illustration with no effect on the functionality of the containing document as a whole. In fact, this is one of the primary strengths of PostScript language document descriptions: the ability to completely merge documents from different sources at the final *printing* stage.

3. POSTSCRIPT LANGUAGE COMMENTS

The comment conventions are divided roughly into the following three areas:

- Structure Comments
- Resource Requirements
- Query Conventions

Structure Comments are used to delimit the various structural components of a PostScript language page description, including its *prologue*, *script*, and *trailer*, and where the page breaks fall (if there are any). These comments have been expanded in this version to include document and page setup information, and to 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. In particular, embedded fonts files, and procedure definitions will have begin/end comment constructs around them to facilitate removing them from the print job (if necessary) or re-structuring the document. *Resource Requirements* are comments used to specify resources which are required by the PostScript language page description, but which have not been included within its text (such as prologues, fonts and included files). These requirement conventions may also specify other document requirements, which may vary from a particular stock form to a specific paper color or collating order. Requirements for individual printer features such as paper trays will also fall into this category. *Query comments* are used to delimit parts of a PostScript language program which query the current state or characteristics of a printer, including the availability of prologues, fonts, files, virtual memory and any printer-specific features and enhancements.

Any or all of these conventions may be used within a single document description, but typically some subset of them will be used consistently within a particular printing environment. The Structuring Conventions have been designed with maximum flexibility in mind, and with a minimum amount of interdependency between the conventions themselves. Thus one may employ only Structure Conventions in an environment where the presence of a spooler may not be guaranteed, or may freely use only the Resource Requirement comments on a highly spooled network. See the section on **Using the Document Structuring Conventions** for more information.

4. CONFORMING FILES

A PostScript language document file is said to be *conforming* if it obeys a proper subset of these structuring conventions. In addition, there are several other constraints which apply to conforming PostScript language documents that will allow document handling software to deal reasonably with the document files. They may be thought of as "promises" that the document description observes certain standard usage and structuring constraints. Primarily these boil down to avoidance of “system-level” PostScript language operators which may cause attempts at processing the documents to fail (see **PostScript language operators to avoid**, *at the end of this section)*. If these structuring conventions are employed, care should be taken to use them correctly and in accordance with their intended goals. *Failure to do so may result in unexpected behavior of document files within some document handling systems.* A simple example of this is a document file which results in an execution error when the pages are re-ordered by a document spooler. This can result simply from the improper employment of the **%%Page:** comment convention.

Here are some general constraints which apply to all conforming PostScript language documents:

- **prologue and script:** When a document is divided into a prologue and script (with the **%%EndProlog** comment and/or the **%%BeginProcSet** comments), there must be nothing *executed* in the prologue (make only definitions) and *no definitions* in the script. The former of these is the more important; the prologue should always be such that it can be removed from a document and downloaded only once (permanently) into the printer. All subsequent documents that are downloaded with this prologue stripped out should still execute correctly.

- **%%Page:** If this comment is used within the body of a document, then the pages *shall not* have any inter-dependencies. Each page may rely on certain PostScript language operations having been defined by the document prologue, but it is not acceptable to have any “state” set in one page of a document that is relied upon by another page in the document. The reason for this is to allow document managers to physically rearrange the document’s pages without affecting the execution of the document description, or even to print different pages in parallel on more than one printer. It is better to *omit* the **%%Page** comment than to employ it without conforming to this constraint.

- **line length:** To provide compatibility with a large body of existing software, a conforming PostScript language document description will 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) and this level of conformance should not pose a problem for software development. Any document structuring comment which needs to be continued on another line to avoid violating this convention should 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 which provide a list of font names (such as **%%DocumentFonts**). Here is an example of its use:

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

- **line endings:** Lines may be terminated with any combination of carriage return or linefeed characters (decimal ASCII 13 and 10, respectively).

- **showpage** If the **showpage** operator is used with **save** and **restore**, the **showpage** shall occur *after* the page-level **restore** operation. The motivation for this is to be able to handily redefine the showpage operator to have side effects in the printer VM such as maintaining page counts for printing *n-up* copies on a single sheet of paper. If the **showpage** is executed within the confines of a page-level **save/restore**, then attempts to redefine showpage to perform extra operations will not work as intended.

- **copypage** If multiple copies of a page are needed, this should be specified by using the #copies convention. The showpage operator in all PostScript interpreter implementations will check for the existence of the name **#copies** in the current dictionary context, and will print as many copies of the page as this name reflects. A typical use of this is as follows:

```
/#copies 3 store
showpage
```

- **PostScript language operators to avoid.** There are some PostScript language operators that are intended for system-level jobs which are not appropriate in the context of a conforming document description. A restriction is placed on their use subject to very careful consideration: a document may be said to be *non-conforming* if any of these operations are used, but if they are used carefully for a very specific reason, they may not actually disturb reasonable handling by document managers. Operations listed with an asterisk (*) should especially be avoided. Others should be used only with extreme care. The risks of using these operators involve either rendering a document *device-dependent* or to unnecessarily inhibit constructive post-processing of document files for different printing needs (for example, embedding one PostScript language document within another). Here is a list of PostScript language operators that should be avoided by conforming document files:

exitserver*	quit*	erasepage*	copypage*
initgraphics*	grestoreall*	initmatrix*	initclip*
framedevice	banddevice	nulldevice	setdevice
setsccbatch	setmatrix	setscreen	settransfer

Note: There is a distinction between a PostScript language *document* and a *print job* (or *session*). A *document* is a data representation that may be transmitted, edited, stored, spooled, or otherwise processed. A print job is an active session with a PostScript printer, which may entail querying the printer, dynamically including (or removing) resources or fonts, even reordering the pages, and may or may not preserve the document exactly as it was originally stored. This is an important distinction, because a *conforming* document is just a document file, and in that context may be asked to conform to use constraints that are not appropriate for an actual PostScript language print job (for instance, the restriction on use of device setup procedures). *This affects primarily programs which act as both document composition programs and as printing managers.* If a job is actually printed (a connection is opened with a printer), then the printing session will need to perform certain document management tasks like querying the printer, but if the PostScript language file is saved to disk instead of being sent to the printer (a useful feature of any document manager) then the file should conform to the Document Structuring conventions (which may mean stripping out all queries, device-dependent PostScript language code, etc.).

4.1 PARSING RULES

Here are a few explicit rules that should apply when parsing these comments (and which should be observed when creating conforming documents). There are also some general rules specified in the previous section on conforming documents that should be followed.

- Many comments have a colon separating the comment from its arguments. This colon is not present in all comments (witness **%%EndProlog**) and should be considered part of the comment itself 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. Unfortunately, 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).

- Comments are case-sensitive, as are all of the arguments following a comment.

- The character set for comments is limited to printable ASCII characters. The comments themselves will only contain alphabetic characters (with the exception of the **?** used to introduce query comments). The arguments may include any character valid in the PostScript language character set, especially where procedure names, font names, and strings are represented.

- When looking for the **%%Trailer** comment (or any **(atend)** comments), make sure to allow for *nested* documents. Observe **%%BeginDocument** and **%%EndDocument** comments, as well as **%%BeginBinary** and **%%EndBinary**.

- The header comments can be explicitly terminated by an instance of **%%EndComments** or implicitly by any line that does not begin with **%%**.

- Lines should never exceed 255 characters, and may be terminated with any combination of carriage return and linefeed characters (decimal ASCII 13 and 10).

- The order of some comments within the document is significant, but within a section of the document, they may appear in any order (for example, within the header section, **%%DocumentFonts**, **%%Title**, and **%%Creator** may appear in any order).

4.2 USING THE DOCUMENT STRUCTURING CONVENTIONS

The Adobe Document Structuring Conventions are designed to facilitate communication between document composition systems and network document handlers. To some extent this will be installation-dependent. Ideally, a document composition system should be able to compose a document without regard to available resources (for instance, paper sizes and font availability), and should be able to rely on a document management system at printing time to determine the availability of the resources and to provide the user with reasonable alternatives.

In keeping with this, the Structuring Conventions contain a certain amount of redundancy. In particular (and as an example), there are two philosophically distinct ways in which a paper size might be specified: in one, the document composition system will *trust* its environment to handle the resource requirements appropriately, and will merely specify what its particular requirements are. On the other hand, a document composer may not know what the network environment holds, and therefore will *include* all the necessary resources (and perhaps printer-specific PostScript language code), but will delimit these included resources or code segments in such a way that a document manager may recognize and replace them. It is up to the software implementor to determine which of these methods is appropriate for a given environment. In some cases, both may be used.

These two distinct philosophies for using the Structuring Conventions are mirrored in the comments themselves. Many of the new Structure comments provide *BEGIN* and *END* constructs intended to identify any printer-specific elements of a document. The document itself should thus be impervious to the presence (or absence) of a document manager or spooler. Many of the Requirement Conventions provide a mechanism to *specify* a need for some resource or printer-specific feature, but to leave the invocation of that feature up 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.

4.3 MINIMALLY REQUIRED COMMENTS

In the Version 1.0 conventions, there were several comment conventions that were *required* in order to minimally conform to that version of the specification. These were **%%DocumentFonts, %%EndProlog, %%Page,** and **%%Trailer**. In version 2.0, the rules are being changed somewhat. *There are no longer any* ***required*** *comments*. All the comments are "optional," in the sense that they may not be appropriate in a given situation, and therefore need not be employed. However, if a comment is used, then it must be used according to this specification. For example, the **%%BoundingBox** comment should be employed if it is appropriate (for instance, if the document is to be used as an illustration in another document), but it is not required if the PostScript language file does not make marks on the page. Similarly, the **%%DocumentFonts** comment is not required if no fonts are used. There are no comments which are strictly required since there are so many different types of PostScript language documents possible. The rule is to *make sure to use them correctly* if you use them. The secondary rule is that you should use as many of these structuring conventions as are appropriate, as they will greatly enhance the ability to use the document in many environments, under many document management systems.

4.4 EXISTING VERSION 1.0 STRUCTURE COMMENTS

PostScript language page descriptions conforming to Version 1.0 of the comment conventions look something like the following example:

```
%!PS-Adobe-1.0
%%Title: title
%%Creator: text
%%CreationDate: text
%%For: text
%%DocumentFonts: (atend)
%%Pages: pagecount
%%BoundingBox: LLx LLy URx URy
%%EndComments

%%EndProlog
%%Page: label 1
%%PageFonts: fontname fontname

%%Trailer
```

5. STRUCTURE COMMENTS

What follows in this document is a list of the comment conventions which comprise Version 2.0 of the Adobe Document Structuring Conventions. All of the existing Version 1.0 conventions are included here, both as a reference and because they are technically also part of the full Version 2.0 specification. These extensions are designed to provide the maximum amount of information and flexibility to a document management system, and can be ignored by any software that does not know how to interpret them. The structure comments are grouped into three types of conventions:

- **Header Comments:** these occur once in a document, and may be thought of as a "table of contents."

- **Body Comments:** these may appear anywhere in a document, and generally consist of *BEGIN\END* constructs to delimit specific ingredients of a document file (such as fonts).

- **Page Comments:** these are page-level structure comments which are analogous to the header comments in purpose.

All of the comments are listed in alphabetical order in the Index at the end of this document for reference. Version 1.0 comments are flagged with an asterisk (*).

5.1 HEADER COMMENTS

The first set of Structure Comments are historically called "header" comments. That means that they appear first in a document file, before any of the executable PostScript language code (and before the prologue). In order to simplify a document manager's job in parsing these header comments, there are two rules that apply to them:

- The first rule is that if there is more than one instance of a header comment in a document file, *the first one encountered* is considered to be the truth. The reason for this is to simplify nesting documents one within another without having to remove the header comments.

- The second rule is that *header comments should be contiguous*. That is, if a document manager comes across a line that does not begin with **"%%"**, then it may quit parsing for header comments. This is important, and easy enough to adhere to. The comments may also be *explicitly* ended with the **%%EndComments** convention *(see below)*.

Note that 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.

%!PS-Adobe-2.0

This comment differs from the previous **%!PS-Adobe-1.0** comment only in version number. It indicates that the PostScript language page description fully conforms to Version 2.0 of the Adobe Structuring Conventions. This comment must occur as the *first* line of the PostScript language file.

There are three *keywords* that may follow this comment on the same line. They are designed to flag the entire print job as a particular *type* of job so that document managers may immediately switch gears to some appropriate mode of processing. Currently the following types of jobs are recognized:

- **EPS** files. These are known as "encapsulated PostScript files," and primarily are PostScript language files which produce illustrations. The format is designed to facilitate including these files in other documents. The exact format of these EPS files is described in another document. The version number of the EPS format may also be specified here (see example below).

- **Query** job. This indicates that the entire job consists of PostScript language queries to which replies are expected. See also the discussion of this under the **Query Comments** section of this document.

- **ExitServer** job. This flags a job that will execute the PostScript language **exitserver** command to permanently register the contents in the PostScript language device. This is needed by some document managers to effectively handle these special jobs. See also the discussion under **%%BeginExitServer**.

The format for these comments is as follows:

```
%!PS-Adobe-2.0 EPSF-1.2
%!PS-Adobe-2.0 Query
%!PS-Adobe-2.0 ExitServer
```

%%Title: text*

This comment provides a text title for the document that is useful for printing banner pages and for routing or recognizing documents. This comment is part of the existing Version 1.0 structuring conventions, as the "*" notation indicates.

%%Creator: text*

Indicates the document creator (usually the name of the document composition software).

%%CreationDate: text*

Indicates the time that the document was created. The time is expressed as a text string, and can be in any format. It is meant to be used purely for informational purposes, such as printing on banner pages, and need not be in a standardized form.

%%For: text*

Indicates for whom the document is intended. This 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.

%%Routing: text

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

%%ProofMode: keyword

This comment provides information about the level of accuracy that is required for printing. It is intended to provide guidance to the printing manager for appropriate tactics when error conditions arise or when resource shortages are encountered. There are three modes which are currently defined:

TrustMe
Substitute
NotifyMe

These modes may be thought of as instructions to the printing manager. If the printing manager detects a resource shortage (such as a missing font or paper size), it should take action based on these proof modes.

The **TrustMe** mode means that the printing manager should take *no* special action. The intent is that the document formatting programs (or the user) knows more than the printing manager. For example, fonts may be available on a network font server that the printing manager does not know about. Even with something like **%%IncludeFile**, if the **%%ProofMode** is **TrustMe**, the printing manager should proceed, even if a file cannot be found. The assumption is that the document itself can compensate for the file not being included.

The **Substitute** mode means that the printing manager should do the best it can to supply missing resources with alternatives. This may mean similar fonts (or bitmapped screen fonts), scaling pages (or tiling) when paper sizes are not available, etc.

The last mode, **NotifyMe**, means that the file should not be printed at all if there are any mismatches or resource shortages noted by the printing manager. An example of this might be expensive printing on a color printer. If the correct font is not available, then the user probably does *not* want a default font.

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

%%BoundingBox: LLx LLy URx URy*

This specifies the bounding box that encloses all the marks painted on the virtual page that encompasses all pages of a document. That is, it must be a "high water mark" in all directions for marks made on any page. All four values must be integers; (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*. The specification **(atend)** is permitted, in which case the deferred comment must occur after the **%%Trailer** comment. See the *PostScript Language Reference Manual, Appendix C* for more details on the **(atend)** notation.

%%Pages: integer [page order]*

This comment takes an integer argument that represents the total number of pages that the document will print, and optionally an argument to indicate the page order. The page count should correspond to the number of pages that would be produced by the execution of **showpage** or **copypage**. If the document produces *no* pages (for instance, if it represents an included illustration which does not use **showpage**), then the page count should be 0. The page order argument is intended to help document managers to determine the order in which the pages are presented in the document file, which in turn will allow a document manager to optionally reorder the pages. It will consist of any of three potential values: **-1, 0, or 1**. A value of **-1** is taken to mean that the pages are in *descending (or n-1)* order, a value of **1** means *ascending (or 1-n)* order, and a **0** represents a *special* order (for instance, signature order). The distinction between a page order code of **0** and no page order at all is as follows: in the absence of the page order argument, any assumption can be made about the page order, and any reordering of the pages is permitted; however, if the page order argument is present and **0**, then the pages should be left intact in the order given. See also the **%%Page** comment. The specification '**(atend)**' is permitted.

%%Requirements: keyword keyword ...

This comment is used to indicate document requirements such as duplex printing, hole punching, collating, sorting, or other physical document processing needs. The **keyword** parameter may correspond to a specific printer feature, or it may be used to designate processing requiring "human intervention." A recommended initial set of values for this comment follows:

duplex	simplex	punch3	punch5
colorprinter	fold	staple	

These keywords may be used together in any combination, or others may be added to compensate special document processing needs. It is up to the document manager to ensure that these requirements are met.

Note: For documents that require **duplex** printing, some printers will be able to handle this directly, while other printers may require human intervention to feed the paper. The document management system must make a decision based on this keyword as to whether or not a printer can be found that will support duplex printing (based on the information provided in the Printer Description files or through querying the printer). If not, some reordering of the document pages may be required. In this case, the **%%Pages** comment should be consulted to provide more information about the existing ordering of pages.

5.2 FONT AND FILE MANAGEMENT

Please read the following section carefully. There are several comments which seem to "interact" with one another, especially within the font handling area. Keep in mind that the **%%DocumentFonts** comment is always the *superset* of any other font comments, and should always contain a list of all fonts used by the documents. Other comments comprise special *subsets* of this list which may be used for font management. There are analogous comments for file and prologue management, as well.

%%DocumentFonts: fontname fontname ...*

This comment is already present in Version 1.0 of the comment conventions, but is included here for reference. It indicates that all the fonts listed are used by the print job. In particular, there will be at least one invocation of the **findfont** operator for each of the font names listed. The notation **(atend)** is permitted here, as specified in the version 1.0 format. If **(atend)** is used, then the list *must* be supplied in the **%%Trailer** section of the document, in the same format as it would normally have appeared in the header comments. See also **%%DocumentNeededFonts** and **%%DocumentSuppliedFonts**.

%%DocumentNeededFonts: fontname fontname ...

This comment (if present) will provide a list of PostScript font names that are *required* by the document (and *not* contained within the document file). It is assumed that there will be at least one corresponding instance of the **%%IncludeFont** directive for each font listed in this section. The purpose of this comment is to allow spoolers to make a decision as to whether the file needs to be parsed to provide font files within the body of the document. See also **%%DocumentFonts** and **%%DocumentSuppliedFonts**.

%%DocumentSuppliedFonts: fontname fontname ...

This convention will provide a list of all the font files that have been provided *within the document print file* as downloaded fonts. There will be at least one corresponding **%%BeginFont** and **%%EndFont** pair within the document description for each of the listed font names. See also **%%DocumentFonts** and **%%DocumentNeededFonts**.

%%DocumentProcSets: name version revision ...

This comment provides a list of all the ProcSets referenced within the document. It is similar to the **%%DocumentFonts** comment in use. The notation **(atend)** is permitted here, as specified in the version 1.0 format. If **(atend)** is used, then the list *must* be supplied in the **%%Trailer** section of the document, in the same format as it would normally have appeared in the header comments. The revision field should be taken to be upwardly compatible with procedure sets 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.

%%DocumentNeededProcSets: name version revision ...

This comment indicates that the document requires the ProcSets listed (for more information, see **%%BeginProcSet**. 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 (such as **""** or **0**). These comments are always used with the **%%IncludeProcSet** comment.

%%DocumentSuppliedProcSets: name version revision ...

This comment indicates that the document contains (or supplies) the ProcSets listed. This corresponds to later use of the **%%BeginProcSet** and **%%EndProcSet** comments. The *name*, *version*, and *revision* fields should uniquely identify the ProcSet.

%%DocumentNeededFiles: filename filename ...

Lists the included files needed by a document description. Each file mentioned in this list will appear later in the document as the argument of a **%%IncludeFile** comment.

%%DocumentSuppliedFiles: filename filename ...

Lists the files included in a document description. Each file mentioned in this list will appear later in the document in the context of a **%%BeginFile** and **%%EndFile** comment construct.

NOTE:
There are now header comments which make reference to the fonts, files, or procedure sets either needed or supplied by a document. These must be carefully used to provide the correct information to a document management program. In particular, they have a specific relationship to each other. The ***%%DocumentNeeded*** *comments are intended to help a document manager decide whether further parsing of a document file is necessary to provide these included resources. They should only be used if the corresponding* ***%%Include*** *comment is used. The* ***%%Document-Supplied*** *comments are extra information for spoolers which wish to “cache” the resources, and provide helpful directories of the resources contained within the print file. They are optional, while the* ***%%DocumentNeeded*** *comments are not optional if the* ***%%Include*** *mechanism is employed. Notice that the* ***%%DocumentFonts*** *comment is special. It existed in Version 1.0, and still is valid, but is treated somewhat differently than the others. The reason for this is that a document may use a font which is believes is permanently loaded into the printer (or in ROM), and therefore need not be explicitly loaded by a server through the* ***%%Include*** *mechanism, nor need be supplied in the document. If mention is made of a font in the* ***%%DocumentFonts*** *line which is determined not to be present in the printer, it should be provided by the font manager, and may be loaded at any point prior to its use by the document file (typically these will be downloaded fonts which are downloaded prior to the document file, but which are not permanently resident in the printer.*

NOTE also:
These comments which provide lists of font or file names may grow longer than the 256 character line length convention. In this case it is permissible

(and recommended) to employ the **%%+** *comment to indicate the continuation of a comment on the following line (see above).*

These conventions (since they are *document level* comments) should be present in the header of a document, if they are used. That is, they should appear before the first instance of **%%EndComments**, or the first non-comment line (which is interpreted as an implicit **%%EndComments**).

%%DocumentPaperSizes: size size ...

Lists all the paper sizes needed by this print job. These sizes will correspond to the nominal size names as specified in the Printer Description file format keywords (see the discussion under **%%BeginPaperSize** for more information).

%%DocumentPaperForms: formname formname ...

Lists all the forms needed by this print job. These form names will correspond to notions of particular forms as understood by a document manager. They might, for instance, include company letterhead, tax forms, or other special forms. They will be explicitly called for by each page that requires them through the use of the **%%PaperForm** comment *(see)*.

%%DocumentPaperColors: colorname colorname ...

Indicates particular colors of paper required by a print job. For example, colorname might be **"goldenrod.**" Again, this comment will be supported by corresponding **%%PaperColor** comments for the particular pages requiring these paper colors.

%%DocumentPaperWeights: integer integer ...

Indicates a weight of paper (in *pounds* or other measure) expressed as an integer value. For example, the value might be **20**. There should be at least one instance of the **%%PaperWeight** comment for each weight listed here. See also **%%PaperWeight**.

%%DocumentPrinterRequired: printer product [version revision]

This directs that the PostScript language page description is intended for a particular printer, identified by its network printer name and/or product string and optionally by its version and revision strings, as defined by the printer's PPD file (and as returned by the **product**, **version**, and **revision** PostScript language operators. This may be used to request a particular printer in a highly networked environment where that printer may be more convenient, to override spooler defaults, or perhaps if the PostScript language file itself contains printer-specific elements. This last case should rarely be necessary, as most documents that require particular features of a PostScript printer can provide Requirement Conventions indicating a need for that feature, rather than requiring a particular brand of printer. Then, if other printers are available which have the necessary features, the page may still be printed as desired. Two examples follow:

```
%%DocumentPrinterRequired: "Local Print" (LaserWriter) 23.0
%%DocumentPrinterRequired: "" "(Linotype)"
```

%%DocumentProcessColors: keyword keyword ...

This comment marks the use of process colors within the document. Process colors are defined to be *cyan*, *magenta*, *yellow*, and *black.* These four colors are indicated in this comment by the keywords **Cyan**, **Magenta**, **Yellow**, and **Black**. This comment is used primarily when producing color separations. See also **%%PageProcessColors**. The **(atend)** conventions is allowed.

%%DocumentCustomColors: name name ...

This indicates the use of custom colors within a document. These colors are arbitrarily named by an application, and their CMYK or RGB approximations are provided through the **%%CMYKCustomColor** or **%%RGBCustomColor** comments within the body of the document. The names are specified to be any arbitrary PostScript language string except (Process Cyan), (Process Magenta), (Process Yellow), and (Process Black), which need to be reserved for custom color implementation by applications. The **(atend)** specification is permitted.

%%EndComments*

This indicates an explicit end to the header comments of the document. Any line that does not begin with **"%%"** also constitutes an implicit end of the header comment block.

%%EndProlog*

This comment marks the boundary between the document's prologue and its script. This convention is already widely observed, and should be included in all documents which have a distinct prologue and script. *Note:* breaking a document into a prologue and a script is conceptually important, although not all document descriptions will fall neatly into this model. If your document represents free form PostScript language fragments which might entirely be considered a "script", you should still include the **%%EndProlog** comment, although there may be nothing in the prologue part of the file. This will effectively make the entire document a "script." Be careful, still, in employing the **%%Page** comment, because document managers may make assumptions about page reversal based on these comments. See the discussion of **%%Page** in the section entitled *Page Structure and Requirement Comments* for more information.

5.3 BODY COMMENTS

Body comments may appear anywhere in a document. They are designed to provide structural information about the document file's organization, and should match any related information provided in the header comments section.

%%BeginSetup
%%EndSetup

These comments delimit the part of the document that does device setup for a particular printer or document. There may be procedures for setting page size, invoking manual feed, establishing a scale factor (or "landscape" mode), or other document-level setup procedures. There may also be general initialization code such as executing **save** or setting some aspects of the graphics state. These delimiters are useful for a document manager to be able to detect an entire section of initialization procedures. If present, this comment will fall after the **%%EndProlog** directive but before the first **%%Page** (In other words, it is *not part of the prologue*. It should be in the first part of the script, before any pages are specified.

Note: Specification of some of the page-level comments are permitted within this section of the document, such as **%%PaperColor**. If found in the document setup section, then they are taken to be in effect for the entire document, rather than for a single page. See the discussion under **Page Comments** in the **Resource Requirements** section of this document.

%%BeginDocument: name [version] [type]
%%EndDocument

These comments are used to delimit an *entire file* that is imported as part of another PostScript language document or print job. This comment is required to allow multiple occurrences of the **%%EndProlog** and **%%Trailer** comment in the body of a document. Any existing document file that is embedded within another document file *must* be surrounded by these comments. The **version** and **type** fields are optional, and if used should provide extra information for recognizing specific documents.

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

%%BeginFile: filename
%%EndFile

The enclosed segment resulted from having included a file in the PostScript language stream. The file server component of a document manager may perhaps extract a copy of this file for later use by the **%%DocumentFiles**, **%%PageFiles** or **%%IncludeFile** comments. The file name will usually correspond to the original disk file name on the host system.

%%BeginFont: fontname [printername]
%%EndFont

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 chosen printer already has the font resident), or it may simply keep a copy of it for later use by the **%%IncludeFont** comment. The **fontname** field should 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.

%%BeginProcSet: name version revision
%%EndProcSet

The PostScript language code enclosed by the **%%BeginProcSet** and **%%EndProcSet** comments typically represents some subset of the document prologue. The prologue may be broken down into many sub-packages, or *procedure sets*, 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 in order to manage them separately for a whole family of documents. Note that an entire document prologue may be an instance of a ProcSet, in that it is a body of procedure definitions used by a document description file. (See also the **%%DocumentProcSets, %%IncludeProcSet** and **%%IncludeFile** 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 prologue is stored in the printer. In any case, these fields are used to identify the ProcSet to the document manager. See also the **%%?BeginProcSetQuery** comment convention, with which one may query the PostScript server for the prologue name and version fields.

Notice that this does not replace the **%%EndProlog** comment used in the original (version 1.0) document structuring conventions. It should still be used in the same manner, although now it is optional to provide the additional **%%BeginProcSet** and **%%EndProcSet** comments as well. A document manager may assume that the document prologue consists of everything from the beginning of the print file through the **%%EndProlog** comment, which may encompass several instances of the **%BeginProcSet/%%EndProcSet** comments.

%%BeginBinary: bytecount
%%EndBinary

These comments are designed to provide information about embedded bodies of binary data. When a PostScript language document file is being parsed, encountering raw data can complicate the parsing process immensely. These comments are designed to allow a document manager to effectively ignore anything encapsulated within these comments. In fact, the data may be anything, including hexadecimal representations of binary data or any other data which should be ignored by a document manager. Since a bytecount field is provided, the material may be skipped even if it contains embedded comment strings. If a PostScript language print job embeds binary data, it should be encapsulated within these **%%BeginBinary** and **%%EndBinary** comments so that parsing software can safely ignore it.

Note: In order 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 **%%BeginBinary/%%EndBinary** construct, then those comments will effectively be *part of the data*, which is typically *not* what is desired. In order 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 that it can be skipped correctly. Make sure to allow for carriage returns, if any. In the example below (taken from the *PostScript Language Reference Manual*, there are 131072 bytes of binary data, but the call to the **image** operator is also encompassed within the **%%BeginBinary** and **%%EndBinary** comments. The resulting byte count includes 6 additional bytes, for the string "image" plus the newline character. *Note:* The entire **%%BeginBinary** comment line should be read before acting on the byte count.

```
/picstr 256 string def
25 140 translate
132 132 scale
256 256 8 [256 0 0 -256 0 256] { currentfile picstr readhexstring pop }
```



```
%%BeginBinary: 131078
image
4c47494b3187c237d237b137438374a
213769876c8976985a5c98767587575 % 131072 bytes binary
%%EndBinary
```

%%BeginPaperSize: sizename
%%EndPaperSize

These comments delimit a PostScript language sequence that will invoke a particular paper size. This comment is provided specifically for environments where simplicity is needed; that is, where the full functionality and generality of the Printer Description files keyword replacement mechanism is not necessary. The **%%BeginFeature** and **%%Feature** comments will allow more specific control over invoking features as found in the Printer Description files. For the purposes of **%%BeginPaperSize** and **%%PaperSize** (see below under *Resource Requirements*), any valid method of securing that particular paper size is acceptable (whether by invoking a particular input bin, a paper tray, or by setting the page size). The appropriate keyword for the nominal paper size should be used as specified in the Printer Description file. Examples of these are: **"Letter, Legal, A3, A4, A5, B4, B5, Ledger, Statement, Tabloid.**" An example follows:

```
%%Page: 1 1
%%BeginPaperSize: Ledger
 statusdict begin ledgertray end
%%EndPaperSize
```

%%BeginFeature: featuretype [option]
%%EndFeature

These comments are used to delimit a PostScript language code segment that will invoke a particular feature on a PostScript printer as specified in the Printer Description File format (described in another document). The **featuretype** will correspond to one of the keywords in the Printer Description file, and the **"featuretype option"** sequence should be exactly as found in the Printer Description file, in order to cooperate effectively with these conventions. The **%%BeginFeature** and **%%EndFeature** comments should delimit any PostScript language fragments that invoke a printer-specific feature on a printer. 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. This is, in a sense, the inverse of the **%%Feature** comment (see the *Resource Requirements* section of this document), which indicates that the specified printer feature needs to be invoked by the document manager at that position in the print file. The **%%Begin/EndFeature** comments are used when the printer feature is being explicitly invoked for a particular printer, and are intended to convey information about this decision to the document manager. This comment can be used instead of using the **%%BeginPaperSize** comment (or even in addition to, if nested properly). Three examples follow. The first example illustrates a PostScript language file in which an assumption is made about the presence of a file system. If a document manager were to send this document to a printer which did have a file system, it might actually replace the encapsulated “false” with the value of that feature (from the Printer Description file) for the chosen printer:

```
% example 1
%%BeginFeature: *FileSystem
 false
%%EndFeature
{ %ifelse
  (logfile) (w) file dup (message) writestring closefile }{ %else
  (message) print flush
} ifelse
```

```
% example 2
%%BeginFeature: *InputSlot Upper
 statusdict begin 1 setpapertray end
%%EndFeature
```

```
% example 3
%%BeginPaperSize: Ledger
%%BeginFeature: *PaperTray Ledger
 statusdict begin ledgertray end
%%EndFeature
%%EndPaperSize
```

%%BeginExitServer: password
%%EndExitServer

These comments delimit the PostScript language sequence that will cause the rest of the file to be registered *outside* the normal server loop **save/restore** context. This convention is used to flag any code that will set up or execute the **exitserver** instruction, so that it can be caught and removed as necessary by a document manager. It may be used in conjunction with the **%%EOF** requirement convention to pinpoint where an end-of-file indication should be sent by the document manager. *Please see also the discussion under* **%!PS-Adobe-2.0** which can be found in the **Structure Comments** section of this document. PostScript language jobs which employ **exitserver** should be specially flagged by use of the **%!PS-Adobe-2.0 ExitServer** notation. An example of its appropriate employment follows:

```
%!PS-Adobe-2.0 ExitServer
%%CreationDate: Mon Feb 2 10:34:36 1987
%%EndComments
%%BeginExitServer: 000000
  serverdict begin 000000 exitserver
%%EndExitServer
% ...
%%EOF
```

%%BeginProcessColor: keyword
%%EndProcessColor

The **keyword** here is either **Cyan**, **Magenta**, **Yellow**, or **Black**. During color separation, the code between these comments should only be downloaded 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. Extreme care must be taken by the document composition software to correctly control overprinting and knockouts if these comments are employed, since the code may or may not be executed.

%%BeginCustomColor: keyword
%%EndCustomColor

The **keyword** here is any PostScript language string except (Process Cyan), (Process Magenta), (Process Yellow), and (Process Black). During color separation, the code between these comments should 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 on the other three separations. Extreme care must be taken by the document composition software to correctly control overprinting and knockouts if these comments are employed, since the code may or may not be executed.

%%CMYKCustomColor: cyan magenta yellow black *keyword*

This provides an *approximation* to the custom color specified by ***keyword***. 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. These numbers are exactly analogous to the arguments to the **setcmykcolor** PostScript language operator. The ***keyword*** follows the same custom color naming conventions for the **%%DocumentCustomColors** comment.

%%RGBCustomColor: red green blue *keyword*

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

%%Trailer*

This comment should 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 this **%%Trailer** comment. Additionally, any of the document-level structure comments that were *deferred* by using the **(atend)** convention (permissible in **%%DocumentFonts, %%Pages, %%PageFonts, %%BoundingBox**, etc.) should be mentioned in the trailer of the document (*i.e.* after the **%%Trailer** comment).

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

5.4 PAGE COMMENTS

The page-level comments will usually occur once for each page (with the exception of the **%%BeginObject** and **%%EndObject** comments), and provide information about that page's requirements or structure.

%%Page: label ordinal*

This comment is used to mark the beginning of the PostScript language code describing a particular page. It requires two arguments: a page label and a sequential page number. The label may be anything, but the ordinal page number should reflect the position of that page within the body of the PostScript language file (and should start with **1**, not **0**). This convention is widely used and very important. You *must* make sure that your pages do not rely on each other, but only on definitions made in the prologue of the document. A document manager should be able to *physically 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). Please do *not* include these comments if the document does not conform to these use constraints. See also the **%%Pages** comment.

%%PageFonts: fontname fontname ...*

Indicates the names of all fonts used on the current page. The notation **(atend)** is permissible here, but the list must be provided before the next instance of **%%Page** or **%%Trailer**. In other words, it must be supplied by the end of that page.

%%PageFiles: filename filename ...

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 (i.e. there are subsequent instances of the **%%IncludeFile** directive on that particular page).

%%PageBoundingBox: LLx LLy URx URy

This is a body comment, in that it will appear in the body of a document, analogous to the **%%PageFonts** comment. It specifies the bounding box that encloses all the marks painted on that particular page (not the whole document—see the **%%BoundingBox** convention). All four values must be integers; (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*. The specification **(atend)** is permitted, in which case the deferred comment must occur before the next **%%Page** or **%%Trailer** comment.

%%PageProcessColors: keyword keyword ...

This comment marks the use of process colors within the page. Process colors are defined to be *cyan*, *magenta*, *yellow*, and *black.* These four colors are indicated in this comment by the keywords **Cyan**, **Magenta**, **Yellow**, and **Black**. See also **%%DocumentProcessColors**. The **(atend)** conventions is allowed.

%%PageCustomColors: name name ...

This indicates the use of custom colors within a document. These colors are arbitrarily named by an application, and their CMYK or RGB approximations are provided through the **%%CMYKCustomColor** or **%%RGBCustomColor** comments within the body of the document. The **(atend)** conventions is allowed.

%%BeginPageSetup
%%EndPageSetup

These directives are analogous to the **%%BeginSetup** comments, except that they will fall within the body of a document, right after a **%%Page** comment. Use them to delimit areas where manual feed is set, margins are established, landscape mode is chosen, particular paper colors are invoked, etc.

%%BeginObject: [name] [code]
%%EndObject

These comments are used to delimit individual graphic elements of a page. In a context where it is desirable to be able to recognize individual page elements, this comment will provide a mechanism to label and recognize them at the PostScript language level. This can be especially useful for a document printing system that may have the capability of only printing selected objects in a document (or on a page). For instance, the **code** field of this comment may be used to represent *proofing levels* for a document *(or color separations)*. Then the printing manager may be requested to "only print those objects with proofing levels less than, say, 4. This can save printing time when proofing various different elements of a document. It can also be useful in systems which allow PostScript language segments to be parsed in and reedited, to allow convenient grouping and categorization of graphic page elements. In a document production system which is highly object-oriented, this comment is strongly recommended.

%%PageTrailer

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.

6. RESOURCE REQUIREMENTS

These comments may appear anywhere in a document, and indicate that the named resource (whether it is a font, a disk file, or other) should be included in the document at the point where the comment is encountered. *These should not appear in the body of a document without a corresponding comment in the header of the document indicating that the files are required by the document as a whole*.

%%IncludeFont: fontname

Indicates that the specified font must be included at this point in the document by the document manager. 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 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.

%%IncludeProcSet: name version revision

This is a special case of the more general **%%IncludeFile** directive. It requires that a PostScript language ProcSet with the given name, version, and revision be inserted into the print file at the current position. If a version numbering scheme is not used, these fields should still be filled with a “dummy” value (such as **""** or **0**). See the **%%BeginProcSet** comment for more information.

%%IncludeFile: filename

Provides a directive that the specified file must be inserted by the document manager at the current position in the document.

%%ExecuteFile: filename

This is much like the **%%IncludeFile** directive except that it specifies that the included file is an *executable* PostScript language file, rather than perhaps a ProcSet or prologue file. This means that in all probability it will contain at least one instance of **showpage**, and means that it can (and should) be wrapped with a **save** and **restore**. This convention should usually be used to include whole PostScript language documents within other documents, and should be used whenever the enclosed file is entirely self-contained (and especially when it can be nested in a **save/restore** context without difficulty). In particular, illustrations and files which have no effect other than to make marks on a page are perfectly suited for the **%%ExecuteFile** convention.

%%ChangeFont: fontname

This comment is being *phased out* as of version 2.1 of this specification. It is not very useful, and is fraught with problems. Please don't use it or feel that you have to support it.

6.1 PAPER COMMENTS

Note: The following comments which request particular paper forms, colors, or sizes *may appear either in the document setup area or on a particular page*. If these comments fall within the document setup section of the document file, then 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, then they should *only be in effect for that page*.

%%PaperForm: formname

This requests a particular form for printing. For example, a page may be printed on company letterhead, on an order form, or other particular form specified within a document handling system. There are no clearly defined values for **formname**; it may vary from one site to another.

%%PaperColor: colorname

The syntax is like **%%DocumentPaperColors**. This comment will appear in the body of a document description after an instance of **%%Page**, and will only remain in effect for that particular page.

%%PaperWeight: integer

The syntax is like **%%DocumentPaperWeights**. This comment will appear in the body of a document description after an instance of **%%Page**, and will only remain in effect for that particular page.

%%PaperSize: sizename

Specifies a particular page size (see **%%BeginPaperSize**) to be set. The **sizename** should be a simple keyword as found in the Printer Description file, and which basically consists of the nominal paper size expressed in lowercase letters (for example: **ledger, legal, a4**). This will only remain in effect for that particular page.

%%Feature: featuretype [option]

This comment specifies the need for a particular printer feature as specified in the Printer Description file format. Its use specifies a *requirement* that must be fulfilled by a document manager before printing (see also the discussion under **%%BeginFeature**). The document file may make the assumption that the **%%Feature** line in the file will be replaced by the appropriate PostScript language fragment from the appropriate Printer Description file, and 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. An example follows. This example will request that the appropriate page dimensions for B4 paper be inserted into the document, and will then use these for a call to the **setpageparams** operator, if it is present. The ***ImageableArea** keyword in the Printer Description file always produces four numbers representing the bounding box of the imageable region of a particular paper size. Two of these numbers can be used as arguments to **setpageparams**, as in the example. The entire invocation is contained within **%%BeginPaperSize / %%EndPaperSize** comments, and may be replaced by entirely different code for a different printer.

```
%%BeginPaperSize: A4
mark 595 842
%%Feature: *ImageableArea A4
 statusdict begin
 /setpageparams where {
   pop 0 0 setpageparams pop pop
 } if
 end
 cleartomark
%%EndPaperSize
```

%%EOF

This comment is used to request that an end-of-file indication be sent to the PostScript device. It may be used, for instance, with the **%%BeginExitServer** comment *(see above)*.

7. QUERY CONVENTIONS

A *query* is defined to be any PostScript language segment that will generate some returned information back to the host across the communications channel that is expected *before* a document can be formatted or created. This might result from the execution of any of the **=, ==, print** or **pstack** operators, for instance. In particular, this definition covers information which is expected back from the PostScript printer for decision-making purposes. This might include the generation of font lists, inquiries as to the availability of resources, printer features, or the like.

All of the 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 should be returned by document managers if they cannot understand or do not support query comments. The value of the default (see below for examples) is entirely application dependent, and can be used by an application to determine specific information about the spooling environment (if any), and to take appropriate default action.

Any print file which embeds PostScript language queries within the job should adhere very carefully to these query conventions, to allow the document to be spooled properly.

7.1 SPOOLER QUERY RESPONSIBILITIES

A spooler which expects to be able to interpret and correctly spool documents conforming to version 2.0 of the Adobe Document Structuring Conventions must, at a minimum, perform certain tasks in response to these query conventions. In general, the spooler must recognize the queries, remove them from the print stream, and send some reply back to the host. If a spooler cannot interpret the query, it is expected to return the value provided as the argument to the **%%?EndQuery** comment (see below). A query can be minimally recognized by the sequence **"%%?Begin"** followed by any number of characters (up to 256 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 will be delimited (minimally) by the sequence **"%%?End"** followed by some keywords and optionally by a colon **":"** (decimal ASCII 58) and the default response to the query (any text through end-of-line). A spooler should make attempts to recognize the full query keyword (like **%%?BeginProcSetQuery**) if it can, but it is obligated at least to respond to any validly formed query.

7.2 QUERY COMMENTS

%!PS-Adobe-2.0 Query

A PostScript language query must be sent as a separate job to the printer in order to be fully spoolable. This means that an *end-of-file* indication should be sent immediately after the query job. A job which is a query job should always begin with this **%!PS-Adobe-2.0 Query** convention, which further qualifies the file as being a special case of a version 2.0 conforming PostScript language file. A query job will contain only query comments, and need not contain any of the other standard structuring conventions. A document manager should be prepared to extract query information from any print file which begins with this comment convention. A document manager should fully parse a query job file until the EOF indication is reached.

Note: It is permissible to include more than one query in a single print job, but not to include queries within the body of a regular print job. It cannot be guaranteed that a print job with embedded queries will be handled properly by a document manager.

%%?BeginQuery: identifier
%%?EndQuery: default

These comments are very general-purpose, and may serve any function that is not adequately covered by the rest of this specification. The reader will notice that the query keywords in this section are very specific, since in order to understand and intelligently respond to a query, a document manager *must* semantically understand the query. Therefore specific keywords like **%%?BeginPrinterQuery** were used. When this generic **%%?BeginQuery** comment is encountered, a spooler may be forced to return the default value. The comment is included primarily for large installations which need to implement specific additional queries that are not covered here, and which will likely implement both the document composition software and the spooling software themselves.

%%?BeginPrinterQuery
%%?EndPrinterQuery: default

This comment delimits PostScript language code that will return 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 should be followed by a newline character, which should match the information in the printer's Printer Description file. This comment may be used also 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
```

%%?BeginVMStatus
%%?EndVMStatus: default

This comment delimits PostScript language code to return the state of the PostScript interpreter's virtual memory. The standard response consists of a line containing the results of the PostScript language **vmstatus** operator.

```
%%?BeginVMStatus vmstatus = = = flush
%%?EndVMStatus: unknown
```

%%?BeginFeatureQuery featuretype option
%%?EndFeatureQuery: default

This query provides information describing the state of some specified printer-specific feature, as defined by the printer's Printer Description file. The **featuretype** field identifies the keyword as found in the Printer Description file specification. The standard response will vary with the feature and is defined by the printer's Printer Description file. In general, the *keyword* associated with the feeture should be returned. In the example that follows, the Printer Description 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

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 available on all implementations. The standard response consists of a line containing the file name, a colon, and either **Yes** or **No**, indicating whether or not the file is present. Look at the example below for the font query, which works the same way.

%%?BeginFontQuery: fontname fontname ...
%%?EndFontQuery: default

This 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 will print a list of values depending on whether or not the font is known to the PostScript interpreter. The font names should be provided on the operand stack by the Document Manager, This is done by simply emitting the names, with leading slash **"/"** characters, before emitting the query itself.

To keep 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 will be **/FontName:Yes** or **/FontName:No**, where each font in the list is returned in this manner. The slashes delimit the individually returned font names, although newlines should be expected (and ignored) between them: A final **'*'** character will follow the returned values.

```
%%?BeginFontQuery: Times-Roman Optima CircleFont Adobe-Garamond
 mark
 /Times-Roman
 /Optima
/CircleFont
/Adobe-Garamond
{ %loop
 counttomark 0 gt { %ifelse
  dup (/) print (1234567890123456789012345678901234567890) cvs print
  FontDirectory exch known { (:Yes) } { (:No) } ifelse =
 }{ %else
  pop exit
 } ifelse
} bind loop (*) = flush
%%?EndFontQuery: Unknown
%%EOF
```

```
/Times-Roman:Yes
/Optima:Yes
/CircleFont:No
/Adobe-Garamond:No
*
```

%%?BeginFontListQuery
%%?EndFontListQuery: default

Provides a PostScript language sequence to return a list of all available fonts. It should consult the FontDirectory dictionary as well as any mass storage devices available to the device. The list need be in no particular order, but each name should be returned separated by a slash **"/"** character. This is normally the way the PostScript **==** operator will return a font name. All white space characters should be ignored. The end of the font list should be indicated by a trailing **"*"** sign on a line by itself (decimal ASCII 42). Here is a look at two valid returns from the query:

```
/Optima/Optima-Bold/Optima-Oblique/Optima-BoldOblique/Courier/Symbol
*
```

```
/Courier
/Symbol
/Times-Roman
*
```

Note:
*In previous versions of this document, it was recommended to use **flush** to separate names into packets. This turns out to result in major performance degradation, and is hereby and subsequently disrecommended.*

%%?BeginProcSetQuery: name version.revision
%%?EndProcSetQuery: default

These comments delimit a ProcSet query. The combination of the **name, version** and **revision** fields should uniquely identify the ProcSet. The standard response to this query will consist of a line containing any of the values **"0, 1, 2"** where a value of **0** means that the ProcSet file is *missing*, a value of **1** means that the ProcSet is *present and OK*, and a value of **2** indicates that the ProcSet is *present but is an incompatible version*.

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

EXAMPLE 1: QUERY COMMENTS

This is an example of a *query job*. It is a separate PostScript language job that is sent before a print file is generated, its purpose being to obtain information about the state of the printer that can be used to generate the print file. Notice the **Query** keyword on the first line of the file, and the use of the query comments.

```
%!PS-Adobe-2.1 Query
%%Title: Query job to find out if some fonts exist
%%?BeginFontQuery: Palatino-Roman Palatino-Bold
mark
/Palatino-Roman
/Palatino-Bold
{
counttomark 0 gt {
dup (/) print (1234567890123456789012345678901234567890) cvs print
FontDirectory exch known { (:Yes) } { (:No) } ifelse =
}{
pop exit
} ifelse
} bind loop (*) = flush
%%?EndFontQuery: Unknown
%%EOF
% send the appropriate EOF indication, depending on protocol
```

EXAMPLE 2: STRUCTURE COMMENTS

This is an example of an actual print file generated by a document manager. It is a representative document with a short prologue and script, a special paper size, and a downloaded font. It does not employ any of the **Requirement Conventions**, and will function properly with no intervention from a document manager or spooler. Since it uses **setpageparams**, it will only run on a PostScript language device that supports this operator unless a spooler changes the PostScript language code for that paper size.

```
%!PS-Adobe-2.1
%%Title: Example print file
%%Creator: /usr/local/bin/emacs
%%CreationDate: Mon Dec 1 09:24:58 1986
%%For: Software Developers
%%BoundingBox: 0 0 612 792
%%Pages: 1
%%DocumentFonts: Palatino-Roman Special
%%DocumentSuppliedFonts: Special
%%DocumentProcSets: /usr/lib/ps/draw.prologue 1.00 0
%%DocumentPaperSizes: ledger
%%EndComments
%%BeginProcSet: "/usr/lib/ps/draw.prologue" "(1.00)" ""
%!
% draw.prologue  Mon Dec 1 09:56:09 1986
/VERSION (1.00) def
/S /save load def
/RS { restore save } bind def
/R /restore load def
/g /setgray load def
/m /moveto load def
/s /show load def
/l /lineto load def
/c /closepath load def
/f { findfont exch scalefont setfont } bind def
```

```
%%EndProcSet
%%EndProlog

%%BeginSetup
%%BeginFeature: *PageSize Ledger
statusdict begin
792 1224 0 0 setpageparams
end
%%EndFeature
%%EndSetup
%%BeginFont: Special
%!
% downloaded font file for the "Special" font
70 dict begin
/FontName /Special def
/etc (more definitions) def
currentdict end
dup /FontName get exch definefont pop
%%EndFont

%%Page: one 1
%%PageFonts: Palatino-Roman Special
%%PaperColor: buff
S
100 100 m
147.5 318.0 l
301.7 77.0 l
0.9 g
c fill
RS
110 110 m
24 /Palatino-Roman f
(Example) s
40 500 m
24 /Special f
(Special Stuff) s
R
showpage
%%Trailer
```

EXAMPLE 3: RESOURCE REQUIREMENTS

This example shows use of the Requirement Conventions. This file is *dependent* on a document manager or spooler to supply it with additional resources. This kind of file is common in large, distributed networks where print spooling is taken for granted and resource management may be centralized.

```
%!PS-Adobe-2.1
%%Title: Example print file
%%Creator: /usr/local/bin/emacs
%%CreationDate: Mon Dec 1 09:24:58 1986
%%For: Software Developers
%%BoundingBox: 0 0 612 792
%%Pages: 1
```

```
%%DocumentFonts: Palatino-Roman Palatino-Bold Sonata
%%DocumentNeededFonts: Sonata Palatino-Roman
%%+ Palatino-Bold
%%DocumentNeededProcSets: "draw.prologue" (1.00) 0
%%DocumentPaperSizes: ledger
%%DocumentPaperColors: buff
%%Requirements: duplex punch3
%%EndComments

%%IncludeProcSet: "draw.prologue" (1.00) 0
%%EndProlog

%%BeginSetup
%%Feature: *PageSize Ledger
%%EndSetup

%%Page: one 1
%%PageFonts: Palatino-Roman
%%PaperColor: buff
S
100 100 m
147.5 318.0 l
301.7 77.0 l
0.9 g
c fill
RS
110 110 m
%%IncludeFont: Palatino-Roman
 24 /Palatino-Roman f
(Example) s
R
showpage
%%Page: two 2
%%PageFonts: Palatino-Bold Sonata
%%PaperColor: white
S
%%IncludeFont: Palatino-Bold
 100 100 m
24 /Palatino-Bold f
(Print something in bold) s
RS
%%IncludeFont: Sonata
 24 /Sonata f
gsave (=======) show grestore
(&Q q|372) show
R
showpage
%%Trailer
```

INDEX

% Comments

%!PS-Adobe-2.0 10
%!PS-Adobe-2.0 Query 26
%%?BeginFeatureQuery 27
%%?BeginFileQuery 28
%%?BeginFontListQuery 28
%%?BeginFontQuery 28
%%?BeginPrinterQuery 27
%%?BeginProcSetQuery 18, 28
%%?BeginQuery 27
%%?BeginVMStatus 27
%%?EndFontListQuery 28
%%?EndFontQuery 28
%%?EndPrinterQuery 27
%%?EndProcSetQuery 28
%%?EndQuery 27
%%?EndVMStatus 27
%%BeginBinary 18
%%BeginCustomColor 21
%%BeginDocument 17, 21
%%BeginExitServer 20
%%BeginFeature 19
%%BeginFile 17
%%BeginFont 17
%%BeginObject 23
%%BeginPageSetup 23
%%BeginPaperSize 19
%%BeginProcessColor 20
%%BeginProcSet 14, 17, 23
%%BeginSetup 16
%%BoundingBox 12, 22
%%ChangeFont 24
%%CMYKCustomColor 21
%%CreationDate 11
%%Creator 11
%%DocumentCustomColors 16
%%DocumentFonts 13
%%DocumentNeededFiles 14
%%DocumentNeededFonts 13
%%DocumentNeededProcSets 14
%%DocumentPaperColors 15, 24
%%DocumentPaperForms 15
%%DocumentPaperSizes 15
%%DocumentPaperWeights 15, 24
%%DocumentPrinterRequired 15
%%DocumentProcessColors 16
%%DocumentProcSets 13, 18
%%DocumentSuppliedFiles 14
%%DocumentSuppliedFonts 13
%%DocumentSuppliedProcSets 14
%%EndBinary 18
%%EndComments 16
%%EndCustomColor 20, 21
%%EndDocument 17
%%EndExitServer 20
%%EndFeature 19
%%EndFile 17
%%EndFont 17
%%EndObject 23
%%EndPageSetup 23
%%EndPaperSize 19
%%EndProcSet 17
%%EndProlog 16
%%EndSetup 16
%%EOF 20, 25
%%ExecuteFile 24
%%Feature 19, 25
%%For 11
%%IncludeFile 23, 24
%%IncludeFont 17, 23
%%IncludeProcSet 14, 23
%%Page 22
%%PageBoundingBox 22
%%PageCustomColors 22
%%PageFonts 22
%%PageProcessColors 22
%%Pages 12
%%PageTrailer 23
%%PaperColor 15, 24
%%PaperForm 15, 24
%%PaperSize 25
%%PaperWeight 15, 24
%%ProofMode 11
%%Requirements 12
%%RGBCustomColor 21
%%Routing 11
%%Title 11
%%Trailer 21
%EndDocument 21
%IncludeFile 18
%IncludeProcSet 18
(atend) 12, 21, 22

N

NotifyMe 12

S

Substitute 11

T

TrustMe 11