POSTSCRIPT®

ENCAPSULATED POSTSCRIPT® FILES
Specification
Version 2.0

June 5, 1989
PostScript® Developer Support Group

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

PN LPS5002

ENCAPSULATED POSTSCRIPT® FILES
Specification
Version 2.0

June 5, 1989
PostScript® Developer Support Group
(415) 961-4111

This document specifies the format required for import of Encapsulated PostScript (EPS) Files into an application. This specification suggests a standard for importing PostScript language files in all environments, and contains specific information about both the Macintosh® and MS-DOS environments. This format conforms to Adobe Systems' Document Structuring Conventions, Version 2.0.

The rules that should be followed in creating importable PostScript language files are a subset of the structuring conventions proposed by Adobe Systems Incorporated; refer to the *PostScript Language Reference Manual*, Appendix C, and *Document Structuring Conventions*, version 2.0, available from Adobe Systems. Files must also be "well-behaved" in their use of certain PostScript language operators, manipulation of the graphics state, and manipulation of the PostScript interpreter's stacks and any global dictionaries. These conventions are designed to allow cooperative sharing of files between many systems using the PostScript language.

Fundamentally, an EPS file is a standard PostScript language file with a bitmap screen preview included optionally in the format. The purpose of an EPS file is to be included into other document makeup systems as an illustration, and the screen representation is intended to aid in page composition. The bitmap is normally discarded when printing, and the PostScript language segment of the file is used instead. Typically any manipulation of the screen image that is performed by the user (such as scaling, translating, or rotation on screen) should be tracked by the page layout application and an appropriate transformation should precede the EPS file when it is sent to the printer.

1. EPS FILE FORMAT GUIDELINES

An EPS file should conform to at least Version 2.0 of the Adobe Document Structuring Conventions. This does not explicitly require any of the structuring comments to be employed, but if used, they should be in accordance with that specification. Additionally, an EPS file is required to contain the %%BoundingBox comment, and is required to be "well-behaved" (see pages 3-4). An EPS file may optionally contain a bitmap image suitable for WYSIWYG screen display, as discussed herein.

The structure of an EPS file is marked by PostScript language comments, according to the *PostScript Document Structuring Conventions*. These are covered briefly here for reference. Structuring comment lines must begin with "%!" or "%%" and terminate with a *newline* (either return or linefeed) character. EPS file conventions require that a comment line be no longer than 256 bytes. A comment line may be continued by beginning the continuation line with `"%%+"`. The EPS file should begin with a header of structuring comments, as specified in the PostScript Structuring Conventions.

2. REQUIRED PARTICIPATION

In order to support Encapsulated PostScript files effectively, some cooperation is required on the parts of those who produce EPS files and those who use EPS files (typically by including them into other documents).

2.1 WHEN PRODUCING EPS FILES

There are certain required comments and several recommended ones that must be provided in the EPS file. These are detailed in Section 3. The file must also be a single page (not a multiple-page document) and must be a *conforming* PostScript language document. Conformance requirements are mostly detailed here, but for the full specification, please refer to the *Document Structuring Conventions* from Adobe Systems.

2.2 WHEN READING AND USING EPS FILES

When including an EPS file into your document, you should basically think of that piece of code as having been generated by your program. After all, that is what all programs (and users) who encounter your print file will think. In particular, you must find out enough about the file to intelligently make it part of your document. The only tricky part of this relates to font usage. This is also the most difficult part of this specification to understand. Basically, you just have to figure out what the requirements of the illustration are and incorporate them into your own requirements (pass them downstream). Then all issues of font management are essentially the same as they were before you included the illustration (and are beyond the scope of this document).

As long as you don't remove relevant information from a file, and as long as you update your global view of font usage and resource requirements to reflect those that you just imported, the rest is fairly easy. The intent behind the EPS specification, in fact, is to make the most of cooperation between producers and consumers of PostScript language files so that neither has to do much, but the combined advantage is great.

3. REQUIRED COMMENTS

The first comment in the header (and the first line in the file) should be the version comment:

%!PS-Adobe-2.0 EPSF-2.0

This indicates to an application that the PostScript language file conforms to this standard. The version number following the word **"Adobe-"** indicates the level of adherence to the standard PostScript Document Structuring Conventions. The version number following the word **"EPSF"** indicates the level of EPSF-specific comments.

The following comment must be present in the header; if it is not present then an importing application may issue an error message and abort the import:

%%BoundingBox: LLx LLy URx URy

The values are in the PostScript default user coordinate system, in points (1/72 of an inch, or 0.3527 mm), with the origin at the lower left corner. The bounding box must be expressed in default user coordinate space. This seems to be a big question among implementors of this specification. Regardless of the coordinate system in which your application operates, here is a foolproof way of determining the correct bounding box: *print* the page, get out a point ruler, and *measure* first to the lower left corner, then to the upper right corner, using the lower-left corner of the physical paper as your origin. This works because it measures the end result (the marks on the page), and none of the computation matters.

4. OPTIONAL COMMENTS

The following header comments are strongly recommended in EPS files. They provide extra information about the file that can be used to identify it on-screen or when printing.

%%Title: included_document_title
%%Creator: creator_name
%%CreationDate: date_and_time

The **%%Creator**, **%%Title**, and **%%CreationDate** comments may be used by an application or spooler to provide human-readable information about a document, or to display the file name and creator on the screen if no bitmapped screen representation was included in the EPS file.

%%EndComments

This comment indicates an explicit end to the header comments, as specified in the structuring conventions.

4.1 HOW TO USE THESE COMMENTS (PHILOSOPHY)

All of the comments in EPS files provide information of some sort or another. Exactly how you use this information is up to you, but you are encouraged not to reduce the amount of information in a file (when you import it or include it, for example) by removing or altering comments. In general, the comments tell you what fonts and files are used, and where. Not everybody cares about these things, but if you do care, then the information is available.

The whole issue of Encapsulated PostScript files is that they are “final form” print files that may be far from the printer that they will actually be imaged on. If they have specific needs, particularly in terms of font usage, these needs must be carefully preserved and passed on downstream, and the program that actually prints the composite document must take pains to make sure the fonts are available at print time.

Any piece of software that generates PostScript language code is potentially both a consumer and a producer of Encapsulated PostScript files. It is probably best not to think that you are at either end of the chain. In particular, if you import an Encapsulated PostScript file, integrate it into your document somehow, and then go to print your document, you are responsible for reading and understanding any of the font needs of the EPS file you imported. These should then be reflected in your *own* font usage comments. If the illustration on page *3* uses the Bodoni font but the rest of your document is set in Times, suddenly your document now also uses the Bodoni font (you included the illustration, after all). This should be reflected in the outermost **%%DocumentFonts** comments and any other appropriate ones.

4.2 FONT MANAGEMENT COMMENTS

If fonts are used, the following two comments (which are defined in version 2.0 of the PostScript Document Structuring Conventions) should be included in the header of the EPS file. The **%%IncludeFont** and **%%Begin/%%EndFont** comments should be thought of as inverses of one another. That is, if you encounter an **%%IncludeFont** comment and actually include a font file at that point, you should enclose the font in **%%BeginFont** and **%%EndFont** comments. Conversely, if you see fit to remove a font from a print file (one that presumably had been delimited with comments), you should always replace it with an **%%IncludeFont** comment rather than completely stripping it. This guarantees the reversibility of your actions.

%%DocumentFonts: font1 font2 ....
%%+ font3 font4

The **%%DocumentFonts** comment provides a full list of all fonts used in the file. Font names should refer to *non-reencoded* printer font names and should be the valid PostScript language names (without the leading slashes). An application that imports an EPS file should be responsible for satisfying these font needs, or at least updating its own **%%DocumentFonts** list to reflect any new fonts.

%%DocumentNeededFonts: font1 font2 ....

The **%%DocumentNeededFonts** comment lists all fonts that are to be included at specific points within the EPS file as a result of the **%%IncludeFont** comment. These fonts must also be listed in the **%%DocumentFonts** comment, but an application may or may not pre-load these at the beginning of the job. The responsibility should be taken, however, by any program that thinks it is actually printing the file, to make sure the fonts requested will be available when the file is printed. This may mean that the individual **%%IncludeFont** comments may be satisfied and the fonts placed in-line, or they may simply be ignored, if the fonts are determined to be already available on the printer. As a third possibility, there may be enough memory to download all the fonts in front of the job and avoid processing the individual requests. This **%%DocumentNeededFonts** comment provides foreshadowing of the **%%IncludeFont** comments to follow, to give printing managers enough information to make these choices intelligently.

%%IncludeFont: fontname

The **%%IncludeFont** comment signals to an application that the specified font is to be loaded at that precise location in the file. It is analogous to the familiar **#include** syntax in the C language. An application should load the specified font regardless of whether the same font has been loaded already by other preceding **%%IncludeFont** comments, since the font may have been embedded within a PostScript language **save** and **restore** construct. However, if the font is determined to be available *prior* to the entire included EPS file (for instance, it may be in ROM in the printer or might have been downloaded prior to the entire print job) the **%%IncludeFont** comment may be ignored by printing manager software.

When an application satisfies an **%%IncludeFont** request, it should *always* bracket the font itself with the **%%BeginFont** and **%%EndFont** comments.

A font that is wholly contained, defined, and used within the EPS file (a downloaded font) should be noted in the **%%DocumentFonts** comment, but *not* the **%%DocumentNeededFonts** comment. The font should follow conventions listed in the Document Structuring Conventions in order to retain full compatibility with print spoolers.

%%BeginFont: fontname
%%EndFont

The **%%BeginFont** and **%%EndFont** comments bracket an included downloadable font. The **fontname** is the simple PostScript language name for the font. These fonts may be stripped from the included file if they are determined to be available (but should be replaced by an **%%IncludeFont** comment).

4.3 FILE MANAGEMENT COMMENTS

%%IncludeFile: filename

This comment, which can occur only in the body of an EPS file, allows a separate file to be inserted at any point within the EPS file. The file might not be searched for or inserted until printing actually occurs, so user care is required to ensure its availability. If it is used, the %%DocumentFiles comment should be used as well. See the Structuring Conventions for more information.

%%BeginFile: filename
%%EndFile

The **%%BeginFile** and **%%EndFile** comments bracket an included file. They are the "inverse" of the **%%IncludeFile** comment. The **filename** is evaluated in the context of the local file system. These files may *not* be stripped from the included file at print time, because they undoubtedly contain executable code. However, they may be temporarily removed, or "factored out" to save space during storage. They should always be replaced by the **%%IncludeFile** comment**.**

4.4 COLOR COMMENTS

%%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. 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.

%%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 actually 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.

5. “WELL-BEHAVED” RULES

An application should encapsulate the imported EPS code in a save / restore construct, which will allow all printer VM (memory) to be recovered and all graphics state restored. Since the code in the imported EPS file will be embedded within the PostScript language that an application will generate for the current page, it is necessary that it obey the following rules, in order to keep from disrupting the enclosing document:

5.1 OPERATORS TO AVOID

The following PostScript operators should not be included in a PostScript language file for import; the result of executing any of these is not guaranteed (see the PostScript Document Structuring Conventions for more on this):

grestoreall	initgraphics	initmatrix	initclip
erasepage	copypage	banddevice	framedevice
nulldevice	renderbands	setpageparams	note
exitserver	setscreen*	settransfer*	

5.2 THE ‘SETSCREEN’ AND ‘SETTRANSFER’ OPERATORS

The **setscreen** operator is troublesome when one file is included within another. **setscreen** is a system-level command that is appropriate for changing the halftone machinery to compensate for marking engine tendencies, but when used for “special effects” can cause problems. For EPS files, the **setscreen** and **settransfer** operators are permitted only under restricted terms.

THE ‘SETTRANSFER’ AND ‘SETCOLORTRANSFER’ OPERATORS

The **settransfer** operator changes the gray-level and color response curves over the interval from **0** to **1**. There are two basic uses of it: to invert an image (typically flipping blacks and whites, less often colors), or to adjust the response curve for a particular output device.

The best (and required) approach for using **settransfer** is to combine your function with the existing one. Here is the recommended way to do this:

```
{ dummy exec  1 exch sub } dup 0 currentransfer put settransfer
```

In this example, the desired transfer function is the code **1 exch sub**. The **dummy exec** essentially executes the existing transfer function before executing the new code. The name **dummy** is replaced by the actual procedure body from the existing transfer function through the **put** instruction. The result is conceptually equivalent to this:

```
{ { original proc } exec 1 exch sub } settransfer
```

This approach is better than “concatenating” procedures because it does not require the existing transfer function to be duplicated (consuming memory).

5.3 THE ‘SHOWPAGE’ OPERATOR

The **showpage** operator is permitted in EPS files primarily because it is present in so many PostScript language files. It is reasonable for an EPS file to use the **showpage** operator if needed (although it is not necessary if the file is truly imported into another document). It is the including application’s responsibility to *disable* showpage if needed. The recommended method to accomplish this is as follows:

TEMPORARILY DISABLING ‘SHOWPAGE’

```
/BEGINEPSFILE { %def
        /EPSFsave save def
        0 setgray 0 setlinecap 1 setlinewidth 0 setlinejoin 10 setmiterlimit [] 0 setdash
        newpath
        /showpage { } def
} bind def
/ENDEPSFILE { %def
        EPSFsave restore
} bind def

BEGINEPSFILE
  100 300 translate
  .5 .5 scale
  %  include the EPS file here, which may execute showpage with no effect

ENDEPSFILE        % restore state and continue
```

This method will only disable the **showpage** operator during the execution of the EPS file, and will restore the previous semantics of **showpage** afterward. It is the responsibility of the EPS file itself to avoid the operators listed in the previous section that might cause unexpected behavior when imported. They need not be redefined along with **showpage**, although it is permissible to do so.

5.4 STACKS AND DICTIONARIES

All of the PostScript interpreter's stacks (including the dictionary stack) should be left in the state that they were in before the imported PostScript language code was executed. This is normally the case for well-written PostScript language programs, and this is still the best way to keep unanticipated side-effects to a minimum. Please avoid unnecessary **clear** and **"countdictstack 2 sub { end } repeat"** cleanup techniques. If you have accidentally left something on one of the stacks, it is best to understand your program well enough to get rid of it, rather than issuing a wholesale cleanup instruction at the end, which will not only clear your operands from the stack, but perhaps will clear other objects as well.

It is recommended that the imported EPS file create its own dictionary instead of writing into whatever the current dictionary might be. Make sure that this dictionary is removed from the dictionary stack when through (using the PostScript language **end** operator) to avoid the possibility of an **invalidrestore** error. Also, no global string bodies should be changed (with either **put** or **putinterval**).

If a special dictionary (like **statusdict**) is required in order for the imported PostScript language code to execute properly, then it should be included as part of the EPS file. However, it should be enclosed in very specific **%%BeginFeature** and **%%EndFeature** comments as specified in the Document Structuring Conventions. No dictionary should be assumed to be present in the printer, and fonts should be reencoded as needed by the EPS file itself.

5.5 THE GRAPHICS STATE

When a PostScript language program is imported into the middle of another executing program, the state of the interpreter may not be exactly in its default state. The EPS file should assume that the graphics state is in its default state, even though it may not be. An importing application may choose to scale the coordinate system or to change the transfer function to change the behavior of the EPS file somewhat. If the EPS file makes assumptions about the graphics state (like the clipping path) or explicitly sets something it shouldn't (the transformation matrix), the results may not be what were expected.

The importing application is responsible for returning the color to be black, the current dash pattern, line endings, and other miscellaneous aspects of the graphics state to their default condition (as specified in the PostScript Language Reference Manual). This can be done in either of two ways: the initial graphics state can be restored from variables, or the state can be explicitly set:

```
/BEGINEPSFILE { %def
    /EPSFsave save def
    0 setgray 0 setlinecap 1 setlinewidth 0 setlinejoin 10 setmiterlimit [] 0 setdash
    newpath
    /showpage { } def
} bind def

/ENDEPSFILE { %def
    EPSFsave restore
} bind def
```

6. FILE TYPES AND FILE NAMING

APPLE MACINTOSH FILES

The Macintosh file type for application-created PostScript language files is **EPSF**. Files of type **TEXT** will also be allowed, so that users can create EPS files with standard editors, although the Structuring Conventions must still be strictly followed. A file of type EPSF should contain a PICT resource in the resource fork of the file containing a screen representation of the PostScript language code. The file name itself may follow any naming convention as long as the file type is **EPSF**. If the file type is **TEXT**, the extensions **.epsf** and **.epsi**, respectively, should be used for the Macintosh-specific format and EPSI interchange format.

MS-DOS AND PC-DOS FILES

The recommended file extension is **.EPS**. For EPSI files, the extension should be **.EPI**. Other file extensions also can be used, but it will be assumed that these files are text-only files with no screen metafile included in them.

OTHER FILE SYSTEMS

In general, the extension **.epsf** is the preferred way to name an EPS file, and **.epsi** for the interchange format. In systems where lower-case letters are not recognized or are not significant, all upper-case can be used.

7. SCREEN REPRESENTATIONS

The EPS file will usually have a graphic screen representation so that it can be manipulated and displayed on a workstation's screen prior to printing. The user may position, scale, crop or rotate this screen representation, and the composing software should keep track of these manipulations and reflect them in the PostScript language code that is ultimately sent to the printing device.

The exact format of this screen representation is machine-specific. That is, each computing environment may have its own preferred bitmap format, and that is typically the appropriate screen representation for that environment. An interchange representation is specified that should be implemented by everyone, and any environment-specific formats can be supported in addition, as deemed appropriate.

7.1 APPLE MACINTOSH: PICT RESOURCE

A QuickDraw™ representation of the PostScript language file can be created and stored as a PICT in the resource fork of the file. It should be given resource number 256. If the PICT exists, the importing application may use it for screen display. If the *picframe* is transformed to PostScript language coordinates, it should agree with the **%%BoundingBox** comment.

Given the size limitations on PICT images, this may not always agree for large illustrations. If there is a discrepancy, the **%%BoundingBox** always should be taken as the "truth", since it accurately describes the area that will be imaged by the PostScript language code itself. In this situation, applications *producing* the preview PICT must all take the same action so that the importing application knows what to do.

Since it is more important to have a reasonable facsimile of the image than it is to have any particular part of it be high quality, the PICT image should be *scaled* to fit within the constraints of the PICT format. That is, the picture will all be there (it will not be cropped), but it will actually be *smaller* than the real image. The importing application should then scale the PICT to a size which matches the bounding box as expressed in the **%%BoundingBox** comment.

7.2 PC/DOS: WINDOWS METAFILE OR TIFF FILE

Either a Microsoft® Windows Metafile or a TIFF (Tag Image File Format) section can be included as the screen representation of an EPS file.

The EPS file itself has a binary header added to the beginning that provides a sort of "table of contents" to the file. This is necessary since there is not a second "fork" within the file system as there is in the Macintosh file system.

NOTE:
It is always permissible to have a pure ASCII PostScript language file as an EPS file in the DOS environment, as long as it does not contain the preview section. The importing application should check the first three bytes of the file. If they match the header as shown below, the binary header should be expected. If the first two match ***%!****, it should be taken to be an ASCII PostScript language file.*

DOS EPS Binary File Header

Bytes	Description
0-3	Must be hex C5D0D3C6 (byte 0=C5)
4-7	Byte position in file for start of PostScript language code section.
8-11	Byte length of PostScript language section
12-15	Byte position in file for start of Metafile screen representation.
16-19	Byte length of Metafile section (PSize)
20-23	Byte position of TIFF representation
24-27	Byte length of TIFF section
28-29	Checksum of header (XOR of bytes 0-27) NOTE: if Checksum is FFFF then it is to be ignored.

Note:
It is assumed that either the Metafile or the TIFF position and length fields are zero; that is, only one or the other of these two formats is included in the EPS file.

The Metafile should follow the guidelines set forth by the Windows specification. In particular, it should not set the viewport or mapping mode, and it should set the window origin and extent. The application should scale the picture to fit within the **%%BoundingBox** comment specified in the PostScript language file.

8. DEVICE-INDEPENDENT INTERCHANGE FORMAT

This last screen representation is intended as an interchange format between widely varied systems. In particular, the bitmap preview section of the file is very simple and is represented as ASCII hexadecimal in order to be more easily transportable. This format is dubbed Encapsulated PostScript Interchange format, or "EPSI."

This format wins no prizes for compactness, but it should be truly portable and requires no special code for decompressing or otherwise understanding the bitmap portion, other than the ability to understand hexadecimal notation.

It is expected that applications that support EPSF will gradually head toward supporting only two formats: the first is the “native” format for the environment in which the application runs (where the preview section is Macintosh PICT or TIFF or Sun raster files or whatever); the second format should simply be this interchange format. Then files can be interchanged between widely varying systems without each having to know the preferred bitmap representation of the others.

%%BeginPreview: width height depth lines
%%EndPreview

These comments bracket the preview section of an EPS file in Interchange format (EPSI). The **width** and **height** fields provide the number of image samples (pixels) for the preview. The **depth** field provides how many bits of data are used to establish one sample pixel of the preview (1, 2, 4, or 8). An image which is 100 pixels wide will always have 100 in the **width** field, although the number of bytes of hexadecimal needed to build that line will vary if **depth** varies. The **lines** field tells how many lines of hexadecimal are contained in the preview, so that they may be easily skipped by an application that doesn’t care. All the arguments are integers.

8.1 SOME RULES AND GUIDELINES FOR “EPSI” FILES

The following guidelines attempt to clarify a few basic assumptions about the EPSI format. It is intended to be extremely simple, since its purpose is interchange. No system should have to do much work to decipher one of these files, and the preview section is mostly just a convenience to begin with. This format is accordingly deliberately kept simple and option-free.

- The preview section must be after the header comment section but before the document prologue definitions. That is, it should immediately follow the **%%EndComments** line in the EPS file.

- In the preview section, **0** is white and **1** is black, in deference to the majority. Arbitrary transfer functions and “flipping” black and white are not supported.

- The Preview image can be of *any* resolution. The size of the image is determined solely by its *bounding box*, and the preview data should be scaled to fit that rectangle. Thus, the **width** and **height** parameters from the image are *not* its measured dimensions, but simply describe the amount of data supplied for the preview. The dimensions are described only by the bounding rectangle.

- The hexadecimal lines must never exceed 255 bytes in length. In cases where the preview is very wide, the lines must be broken. The line breaks can be made at any even number of hex digits, since the dimensions of the finished preview are established by the **width**, **height**, and **depth** values.

- All non-hexadecimal characters should be ignored when collecting the data for the preview, including tabs, spaces, newlines, percent characters, and other stray ASCII characters. This is analogous to the PostScript language **readhexstring** operator.

- Each line of hexadecimal will begin with a percent sign ('**%**'). This makes the entire preview section into a PostScript language comment, so that the file can be printed without modification.

- If the **%%IncludeFile** or **%%BeginFile / %%EndFile** comments are ever used to extract the preview section from the EPS file, then the **lines** argument to the **%%BeginPreview** comment must be adjusted accordingly. The **lines** value specifies only the number of lines to *skip* if you're not the least bit interested.

- If the **width** of the image is not a multiple of 8 bits, the hexadecimal digits are padded out to the next highest multiple of 8 bits.

EXAMPLE "EPSI" FILE

Here is a sample file showing the EPS Interchange (EPSI) format. The preview section is expressed in user space and the correct comments are included. Remember that there are 8 bits to a byte, and that it requires 2 hexadecimal digits to represent one binary byte. Therefore the 80-pixel width of the image requires 20 bytes of hexadecimal data, which is (80 / 8) * 2. The PostScript language segment itself simply draws a box, as can be seen in the last few lines.

```
%! PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 80 24
%%Pages: 0
%%Creator: Glenn Reid
%%CreationDate: September 19, 1988
%%EndComments
%%BeginPreview:  80  24  1  24
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FF0000000000000000FF
% FF0000000000000000FF
% FF0000000000000000FF
% FF0000000000000000FF
% FF0000000000000000FF
% FF0000000000000000FF
% FF0000000000000000FF
% FF0000000000000000FF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
% FFFFFFFFFFFFFFFFFFFF
%%EndPreview
%%EndProlog
%%Page: "one" 1
 4 4 moveto 72 0 rlineto 0 16 rlineto -72 0 rlineto closepath
 8 setlinewidth stroke
%%Trailer
```