POSTSCRIPT™
Software From Adobe

# Encapsulated PostScript File Format Specification

*Adobe Developer Support*

Version 3.0

1 May 1992


Adobe Systems Incorporated

Corporate Headquarters
1585 Charleston Road PO Box 7900
Mountain View, CA 94039-7900
(415) 961-4400 Main Number
(415) 961-4111 Developer Support
Fax: (415) 961-3769

Adobe Systems Eastern Region
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120
Fax: (617) 273-2336

Adobe Systems Europe B.V.
Europlaza
Hoogoorddreef 54a
1101 BE Amsterdam Z-O, Netherlands
+31-20-6511 200
Fax: +31-20-6511 300

Adobe Systems Japan
Swiss Bank House 7F
4-1-8 Toranomon, Minato-ku
Tokyo 105, Japan
81-3-3437-8950
Fax: 81-3-3437-8968



PN LPS5002

# Contents


1 Introduction 1

2 Guidelines for Creating EPS Files 3
- Required DSC Header Comments 3
- Conditionally Required Comments 5
- Recommended Comments 6
- Illegal and Restricted Operators 6
- Stacks and Dictionaries 6
- Graphics State 7
- Initializing Variables 7
- Ensuring Portability 8
- Miscellaneous Constraints 9

3 Guidelines for Importing EPS Files 9
- Displaying an EPS File 9
- Producing a Composite PostScript Language Program 10

4 File Types and Naming 18
- Apple Macintosh File System 18
- MS-DOS and PC-DOS File System 18
- Other File Systems 18

5 Device-Specific Screen Preview 18
- Apple Macintosh PICT Resource 19
- Windows Metafile or TIFF 19

6 Device-Independent Screen Preview 20
- Guidelines for EPSI Files 21

7 EPS Example 23

**Appendix: Changes Since Earlier Versions** 27

**Index** 29

# Encapsulated PostScript File Format Specification

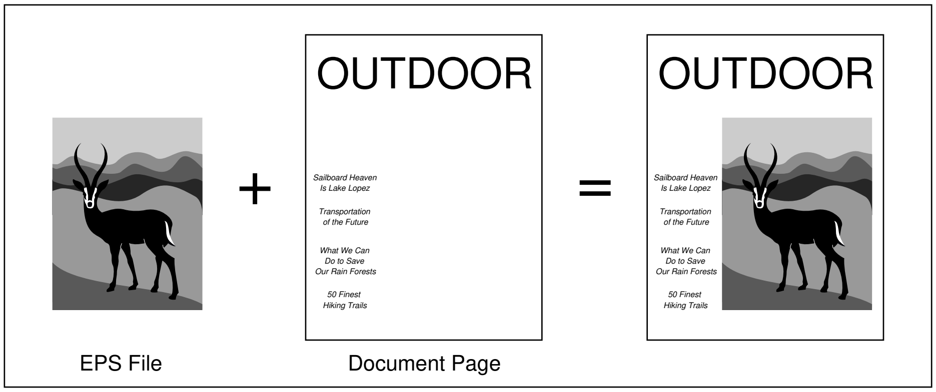
The encapsulated PostScript file (EPSF) format is a standard format for importing and exporting PostScript language files among applications in a variety of heterogeneous environments. This appendix details the format and contains specific information about the Macintosh® and MS-DOS® environments. The EPSF format is based on and conforms to the document structuring conventions (DSC) detailed in the *PostScript Document Structuring Conventions Specification* available from the Adobe Systems Developers Association. Proper use of the document structuring conventions is required when creating a PostScript language file that conforms to the EPSF format.

The main topics of this appendix include creating encapsulated PostScript (EPS) files, importing EPS files into other PostScript language files, and optional screen preview images for EPS files. Finally, a detailed example illustrates the concepts presented throughout this appendix.

## 1 Introduction

An encapsulated PostScript file is a PostScript language program describing the appearance of a single page. Typically, the purpose of the EPS file is to be included, or "encapsulated," in another PostScript language page description. The EPS file can contain any combination of text, graphics, and images, and it is the same as any other PostScript language page description with only a few restrictions. Figure 1 conceptually shows how an EPS file can be included in another PostScript language document.

**Figure 1** *Document with an imported EPS file*




Applications that create conforming EPS files must follow the guidelines in section section 2.*"* There are two required DSC comments, some conditionally required comments, and several programming guidelines to ensure that the EPS file can be reliably imported into an arbitrary PostScript language page description without causing any side effects. An example of a side effect is erasing the page of the importing document or terminating the print job.

Applications that import EPS files must follow the guidelines in section section 3.*"* An application importing an EPS file must parse the EPS file for DSC comments and extract at least the bounding box and resource dependencies of the EPS file. The application should also read and display the screen preview, if present. If there is no screen preview provided in the EPS file, the application must provide an alternate representation and allow the user to place and transform the preview on the screen.
The application must then convert the user's manipulations into the appropriate transformation to the PostScript coordinate system before sending the document to the printer. The application must also preserve its stacks, current dictionary, and graphics state before the imported EPS file is executed.

Note that EPS files are a *final-form* representation. They cannot be edited when imported into a document. However, the imported EPS file as a whole may be manipulated to some extent, including transformations such as translation, rotation, scaling, and clipping.

The device-independent nature of the PostScript language makes it an excellent interchange format. However, it normally requires a PostScript language interpreter to preview an EPS file on screen. Display PostScript systems allow EPS files to be dynamically interpreted, insuring the highest-quality, on-screen preview regardless of scale, rotation, or monitor type. For other environments where the Display PostScript system is not available, the EPS file format allows for an optional screen preview image.

The format of this preview representation varies from system to system. It is typically a Macintosh PICT resource, a TIFF file, or a device-independent hex bitmap. If the EPS file does not provide a preview image, the application that includes the EPS file must provide a representation of the preview, such as a gray box that represents the extent of the EPS file. The end user can use the screen preview to position and size the EPS file in the document.

To support encapsulated PostScript files effectively, some cooperation is required among the applications that *produce* EPS files and those that *use* EPS files. Typically, EPS files are used by importing (or including) them in other documents.

All DSC comments in an EPS file communicate information. How an application uses this information is up to the programmer of the including application. When importing an EPS file, do not reduce the amount of information in the EPS file by improperly removing or altering DSC comments. In general, the comments indicate what resources and language extensions are used, and where they are used in the EPS file. Encapsulated PostScript files are final-form print files that do not know anything about the printer on which they will be imaged. If they have specific resource needs, such as fonts, these needs must be carefully preserved and addressed.

Any application that generates PostScript language programs is potentially both a *consumer* and a *producer* of encapsulated PostScript files. It is probably best not to think that an application is at either end of the chain. If an application imports an EPS file, it is responsible for reading and understanding any of the resource needs of the imported EPS file. These needs must be reflected in the resource usage comments of the composite document the including application creates. For example, if an imported EPS file uses Lithos™, but the rest of the document is set in Times-Roman, then by importing the EPS file, the document now also uses the Lithos font. This fact must be reflected in the composite document's outermost %%DocumentNeededFonts: comment. This concept holds true for the %%DocumentNeededResources:, %%LanguageLevel: and %%Extensions: comments as well.

## 2 Guidelines for Creating EPS Files

To be considered a conforming EPSF version 3.0 file, a file must follow the rules set forth in this appendix, be a *single* page document that fully conforms to the DSC version 3.0 or later (described in the *PostScript Document Structuring Conventions Specifications* available from the Adobe Systems Developers' Association), and include two required DSC header comments.

### 2.1 Required DSC Header Comments

The two required DSC Header comments are

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: llx lly urx ury
```

The first required DSC header comment informs the including application that the file conforms to version 3.0 of the EPSF format as described in this appendix. This is the version comment.

The second required DSC header comment provides information about the size of the EPS file and must be present so the including application can transform and clip the EPS file properly. This is the bounding box comment.

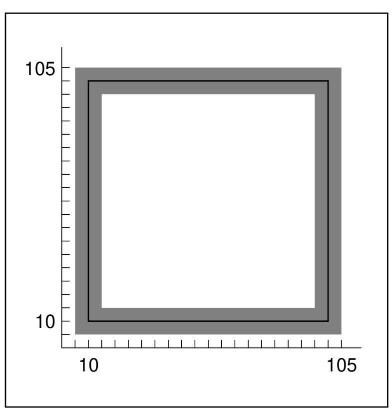
The four arguments of the bounding box comment correspond to the lower-left (*llx, lly*) and upper-right (*urx, ury*) corners of the bounding box. They are expressed in the default PostScript coordinate system. For an EPS file, the bounding box is the smallest rectangle that encloses all the marks painted on the *single* page of the EPS file. Graphics state information, such as the current line width and line join parameters, must be considered when calculating the bounding box. Example 1: shows a minimally conforming EPS file that draws a square with a line width of 10 units.

**Example 1:**

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 5 5 105 105
10 setlinewidth
10 10 moveto
0 90 rlineto 90 0 rlineto 0 -90 rlineto closepath
stroke
```

The marks painted by Example 1:, and how they are positioned with respect to the PostScript coordinate system, are illustrated in Figure 2. If the line width were not considered when calculating the bounding box, the bounding box would be incorrectly positioned by five units on each side of the square, causing the application to incorrectly place and clip the imported EPS file. The bounding box specified for this example is correct.

**Figure 2** *Calculating the correct bounding box*



Regardless of the coordinate system in which an application operates, there is a convenient way to estimate the bounding box: Print the page, then use a point ruler to measure from the lower-left corner of the paper to the lower-left corner of the image. Then measure to the upper-right corner, also using the lower-left corner of the paper as the origin. These two measurements give the bounding box and do not depend on any computation.

## 2.2 Conditionally Required Comments

There are several optional DSC comments that may be conditionally required for a conforming EPS file. These comments must appear in an EPS file if certain features are present—for example, comments to bracket the preview section or to state that a certain language version or language extensions must be present in the interpreter.

The %%Begin(End)Preview comments must bracket the preview section of an EPS file if the preview is represented in the encapsulated PostScript interchange (EPSI) format. See section section 6, " for details and an example of EPSI.

The %%Extensions: comment is required if the EPS file requires a PostScript language interpreter that supports particular PostScript language extensions to print properly. For example, the EPS file may contain CMYK language extension operators and must be sent to a printer that can handle those operators. In such a case, the EPS file must contain either the %%Extensions: CMYK or the %%LanguageLevel: 2 comment.

The %%LanguageLevel: comment is required if the EPS file uses Level 2 features without providing conditional emulation. With this information, the including application can alert the user and avoid any errors that would be generated if the file were sent to a Level 1 printer.

If the EPS file uses language extensions or Level 2 features, and it provides complete emulation of the features in terms of Level 1 operators, the %%Extensions: and %%LanguageLevel: comments are not necessary. See Appendix D of the *PostScript Language Reference Manual, Second Edition* for compatibility and emulation strategies.

If the EPS file requires any fonts, files, forms, patterns, procsets (procedure sets), or any other resources, the appropriate DSC comment must appear in the header comments section of the file. See the *PostScript Document Structuring Conventions Specifications* available from the Adobe Systems Developers' Association.

## 2.3 Recommended Comments

An application or spooler may optionally use the general header comments %%Creator:, %%Title:, and %%CreationDate: to provide information about a document. These header comments are strongly recommended for EPS files.

## 2.4 Illegal and Restricted Operators

There are some PostScript language operators plus **statusdict** and **userdict** operators that are intended for system-level jobs or page descriptions that are not appropriate in an EPS file. In addition to all operators in **statusdict** and the operators in **userdict** for establishing an imageable area, the following operators must not be used in an EPS file:

| | | | |
|---|---|---|---|
| banddevice | exitserver | initmatrix | setshared |
| clear | framedevice | quit | startjob |
| cleardictstack | grestoreall | renderbands | |
| copypage | initclip | setglobal | |
| erasepage | initgraphics | setpagedevice | |

If used properly, the following operators are allowed in an EPS file. However, use of any of these must comply with the rules in Appendix I of the *PostScript Language Reference Manual, Second Edition*. Improper use can cause unpredictable results.

| | | | |
|---|---|---|---|
| nulldevice | sethalftone | setscreen | undefinefont |
| setgstate | setmatrix | settransfer | |

## 2.5 Stacks and Dictionaries

The PostScript interpreter's operand and dictionary stacks *must* be left in the state they were in before the EPS file was executed. The EPS file must not leave objects on either of these two stacks as a result of its execution. All operators placed on the operand stack must be used or removed from the stack with the **pop** operator.

It is strongly recommended that an EPS file make all of it definitions in its own dictionary. This means an EPS file should create its own dictionary or dictionaries instead of writing into the importing application's dictionary. In Level 1 interpreters, the dictionary the importing application uses may not have room for the EPS file definitions. Also, to avoid the possibility of an **invalidrestore** error, make sure the EPS file's dictionary is removed from the dictionary stack using the PostScript language operator **end** when the EPS file has finished using it. Every dictionary that the EPS file places on the dictionary stack with a **begin** operator must be removed from the dictionary stack by the EPS file with a corresponding **end** operator.

*Note* *Do not use the* **clear** *or* **cleardictstack** *operators to clear the stacks in an EPS file. These wholesale cleanup operators not only clear the EPS file's operands and dictionaries from the stacks, they may clear other objects as well.*

The PostScript dictionary lookup mechanism searches the dictionaries that are on the dictionary stack. Bypassing the dictionary lookup mechanism for system-level names is *illegal* in an EPS file. *Do not use the following type of code:*

```
/S systemdict /showpage get def% Illegal EPS code
```

It may cause incorrect results in the including application's PostScript output by overriding the application's redefinitions.

## 2.6 Graphics State

An application importing an EPS file may transform the PostScript coordinate system or alter some other aspect of the graphics state so it is no longer in its default state. This allows the application to change the appearance of the EPS file, typically by resizing, clipping, or rotating the illustration. If the EPS file makes assumptions about the graphics state, such as the current clipping path, or explicitly sets something it shouldn't, such as the transformation matrix (see section section 2.4*"*), the results may not be what were expected.

In preparation for including an EPS file, the graphics state must be set by the including application as follows: current color to black, line caps to square butt end, line joins to mitered, line width to 1, dash pattern to solid, miter limit to 10, and current path to an empty path. Also, if printing to a Level 2 interpreter, overprint and stroke adjust should be set to *false*. An EPS file can assume that this is the default state. It is the responsibility of the application importing the EPS file to make sure that the graphics state is correctly set.

## 2.7 Initializing Variables

It is common for PostScript language programs to use short names, such as x, for variables or procedures. Name-conflict problems can occur if an EPS file does not initialize its variables *before* defining its procedures—in particular, before binding them. In the following example, the variable x is not initialized before being used in the procedure proc1. Because the value of x in the enclosing program happens to be an operator, **bind** causes the name x to be replaced by the operator **lineto** in proc1. This causes a **stackunderflow** error upon execution.

```
%!PS-Adobe-3.0
...Document prolog of including application...
/x /lineto load def       % Application defines x to be lineto
```

```
...More of document prolog and setup...
%%BeginDocument: GRAPHIC.EPS
...Document prolog and setup for EPS file...
/proc1 {                  % Enter deferred execution mode
  /x exch def
  x 4 moveto
  } bind def              % x associated with lineto after bind
4 proc1                   % Execute proc1 and cause error
...Rest of EPS file...
%%EndDocument
...Rest of including application document...
```

In the following example, the EPS file *correctly* initializes the variable x before defining the procedure proc1:

```
%!PS-Adobe-3.0
...Document prolog of including application...
/x /lineto load def       % Application defines x to be lineto
...More of document prolog and setup...
%%BeginDocument: GRAPHIC.EPS
...Document prolog and setup for EPS file...
/x 0 def                  % Initialize variables before defining procs
/proc1 {
  /x exch def
  x 4 moveto
} bind def
4 proc1                   % Execute Proc1
...Rest of EPS file...
%%EndDocument
...Rest of including application document...
```

## 2.8 Ensuring Portability

Although using outside resources, such as fonts, patterns, files, and procsets, is allowed in an EPS file, the most portable files are those that are self-contained and do not rely on outside resources. For example, if an EPS file requires an encoding other than the default encoding for a font, then the EPS file should perform the re-encoding.

EPS files must never rely on procedures that are defined in application- or driver-provided prologs, such as procedures defined in the Apple LaserPrep file. Such definitions might or might not be present, depending on the actions of the enclosing program or previous jobs.

Because EPS files should be portable across heterogenous environments, 7-bit ASCII is the recommended format for data in EPS files. Although binary data is allowed, use caution when producing data that is expected to be portable. The use of binary data may make it impossible to print on some printers across some communication channels. Binary data that has special meaning, such as “flow control” or “marking the end of a file,” can cause file transmission problems in certain communications environments. For example, the control-D character is used as an end-of-file indicator in serial and parallel communications channels. Because this character terminates the job in serial and parallel environments, it is not prudent to produce an EPS file with this character in it.

See Appendix D of the *PostScript Language Reference Manual, Second Edition* for guidelines about how to take advantage of language extensions and Level 2 features while maintaining compatibility with Level 1 PostScript interpreters.

### 2.9 Miscellaneous Constraints

EPS files must not have lines of ASCII text that exceed 255 characters, excluding line-termination characters.

Lines must be terminated with one of the following combinations of characters: CR, LF, CR LF, or LF CR.

CR is the carriage return character and LF is the line feed character (decimal ASCII 13 and 10, respectively).

## 3 Guidelines for Importing EPS Files

This section contains guidelines that should be followed when creating an application that imports EPS files. The first part discusses displaying an EPS file; the second covers producing the PostScript language code for the printer.

This section contains several PostScript language code fragments. A complete code example that implements all of these segments is in section section 7.”

### 3.1 Displaying an EPS File

There are several techniques for including an EPS file in a document. The following scenario is typical:

1. When the user imports an EPS file, the application prompts the user to select the EPS file to be imported.

2. The application opens the selected file and parses it for useful information. If either of the two required header comments is missing, the application should alert the user that the file is not a conforming EPS file and abort the import.

The DSC elementary type (atend) may be used to defer bounding box data to the end of the EPS file. This means an application may need to parse through the %%Trailer comments to obtain the bounding box data.

3. If the version and bounding box comments are found, the application should prompt the user to place the EPS file. It should then display the screen preview. If no preview is provided with the EPS file, the application must provide a representation of the EPS file.

   If the application must create its own representation, a gray box matching the extent of the bounding box with some information in it suffices. The information should at least include the title of the EPS file. This can be obtained from the DSC header comment: %%Title:. Other information, such as %%Creator: and %%Creation-Date:, may also be displayed.

The bounding box comment can be used to help determine scaling factors and the proportions of the illustration. The including application should enable the user to specify a “placement box” to display the screen preview or the application-supplied representation of the screen preview if there is not a preview present in the EPS file.

The bounding box can be used to calculate a ratio that the application can use if the user wants to maintain original proportions while specifying a placement box. Alternately, the application may display the preview full size, and then allow the user to size and place the graphic as desired. Regardless of the method used to display the preview initially, the user should have the option of maintaining the original proportions supplied by the bounding box or distorting the proportions of the EPS graphic.

## 3.2 Producing a Composite PostScript Language Program

The following guidelines must be considered when producing a composite PostScript language program that includes an imported EPS file.

### Use save and restore

An application should encapsulate the imported EPS file in a **save/restore** construct. This allows all VM the EPS file uses to be recovered and the graphics state to be restored.

### Redefine showpage

The **showpage** operator is permitted in EPS files because it is present in so many PostScript language files. Therefore, it is reasonable for an EPS file to use the **showpage** operator, although it is not necessary if the EPS file will only be imported into another document. The application importing the EPS file is responsible for redefining **showpage**. **showpage** may be redefined using the following code segment:

```
/showpage { } def
```

### Prepare the Graphics State

In preparation for including an EPS file, the including application must set the graphics state as follows: current color to black, line caps to square butt end, line joins to mitered, line width to 1, dash pattern to solid, miter limit to 10, and the current path should be set to an empty path. This state can be explicitly set using the following code segment:

```
0 setgray 0 setlinecap 1 setlinewidth
0 setlinejoin 10 setmiterlimit [ ] 0 setdash newpath
```

Also, if printing directly to a Level 2 printer, the overprint and stroke adjust graphics state parameters must be set to *false*. This can be done by conditionally using the following code segment:

```
false setoverprint false setstrokeadjust
```

*Note* *If the application knows that any given parameter of the current graphics state is already in its default state, there is no need to execute the related PostScript language code to reset that parameter.*

### Push userdict

It is recommended that an application importing an EPS file use the **begin** operator to push a copy of **userdict** on top of the dictionary stack. Ideally, the imported EPS file should create its own dictionary, but if it does not, and if the application's dictionary does not have enough room for the EPS file's definitions**,** a **dictfull** error may result when the EPS file makes its definitions. After execution of the EPS file, the application should remove the copy of **userdict** from the dictionary stack by executing the **end** operator.

### Clear the Operand Stack

The application importing the EPS file must leave an empty operand stack for the EPS file. It is reasonable for the EPS file to expect that the entire operand stack be available for its own use. If the entire operand stack is needed and is not available, a **stackoverflow** error may occur. Also, if the operand stack is empty, an EPS file that inappropriately executes **clear** will not cause any problems.

### Protect the Stacks

An EPS file should leave the operand and dictionary stacks as they were before the EPS file was executed. However, this may not always be the case. So before including the EPS file, the importing application should be sure to count the number of objects on the dictionary and operand stacks.

Then, after executing the EPS file, it should make sure the stacks contain the same number of objects as they did before the EPS file was executed. The following code segment shows how to obtain the count of objects on the dictionary and operand stacks:

```
/Dict_Count countdictstack def
/Op_Count count def
```

### Bracket EPS File with Comments

The included EPS file must be bracketed by the %%Begin(End)Document: comments as described in the *PostScript Document Structuring Conventions Specifications* available from the Adobe Systems Devlopers' Association.

### Handle Special Requirements

If either the %%LanguageLevel: comment or the %%Extensions: comment is present in the header comments section of the EPS file, then at print time the application printing the composite file is responsible for assuring that the printer can handle the specified language extensions. If the application determines that the printer does not have the necessary language features to print the document properly, or if the application cannot determine extension availability, the user should be notified and prompted for the appropriate action. Also, if an application has imported an EPS file that requires extensions, the application's output is now dependent on the *same* extensions. This must be reflected in the document's header comment section.

If any %%DocumentNeededResources: or %%DocumentNeededFonts: comments are present in the header comments section of the EPS file, before printing the document the application must be sure the resources are available. If any of the resource requirements cannot be handled, the user must be notified and prompted for an appropriate action. Such an action may involve having the user locate the resource or allowing the user or document manager to reroute the print job to a printer that has the required resources. Also, if an application has included an EPS file that requires these comments, the application's output is now dependent on the same resources. This must be reflected in the document's header comment section.

### Default Coordinate System Transformation

Before including the EPS file in its page description, the importing application must transform the PostScript coordinate system according to the final user placement of the EPS file. The order of the transformation sequence must be:

1. Translate the origin to the new user-chosen origin.

2. Rotate, if the user has rotated the EPS file.

3. Scale, if the user has changed the size.

4. Translate the lower-left corner of the EPS file's bounding box to the user-chosen origin.

Details on transforming the PostScript coordinate system are below. The first example is a simple case in which the user coordinate system matches the default PostScript coordinate system. The second example is a general case transformation from application space to the default PostScript coordinate system.

Figure 3 shows an EPS file and its bounding box superimposed on a target page. The EPS file is shown as it would be drawn if the EPS file were printed without first transforming the PostScript coordinate system. The placement box in the upper-right corner of the page shows where the user chose to place the EPS file.

**Figure 3** *EPS file and placement box*

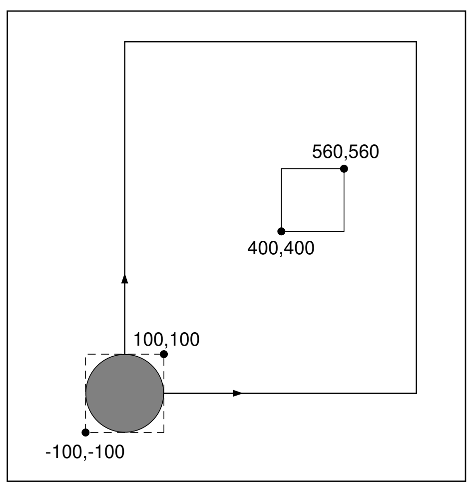
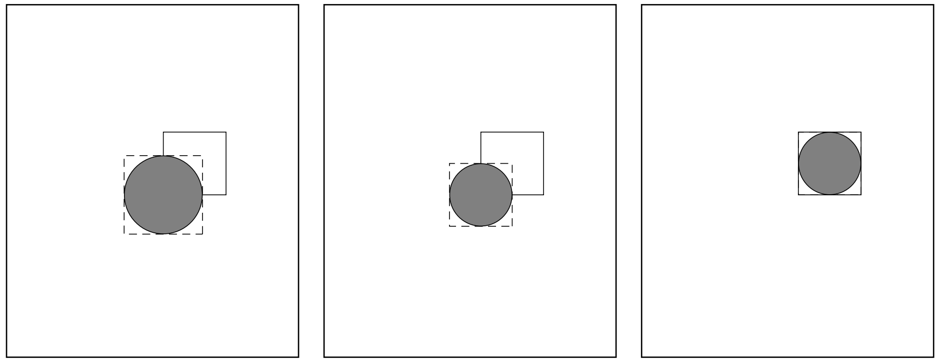



Figure 4 contains three diagrams that show the steps necessary to properly translate and scale the PostScript coordinate system to achieve the user-chosen placement on the page.

**Figure 4** *Transforming the EPS file*



*Translate to new origin* *Scale to fit placement box* *Translate to final position*

Assuming that the bounding box found in the header of the EPS file is %%BoundingBox: -100 -100 100 100, the following PostScript language code fragment properly places the EPS file on the printed page:

```
400 400 translate         % Translate to new origin
.8 .8 scale               % Scale to fit "placement
box"
100 100 translate         % -llx -lly translate
```

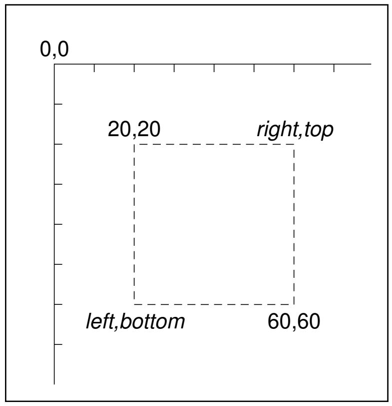
This transformation code must be inserted into the PostScript stream *ahead* of the EPS code being sent to the printer.

Figures H.3 and H.4 and the corresponding PostScript code fragment assume that the application coordinate system matches the default PostScript coordinate system. The following section discusses a more general coordinate system transformation.

### General Coordinate System Transformation

Typically, an application transforms the PostScript coordinate system so the native drawing units of the application space can be used as the operands to the PostScript language operators defining the page. Consider Figure 5, which represents an arbitrary application coordinate system and a placement box for an EPS file.

**Figure 5** *Application coordinate system plus placement box*




To transform the PostScript coordinate system to match the application coordinate system in Figure 5, an application could execute the following code fragment:

```
0 792 translate
1 -1 scale
```

This assumes that each unit of application space is equal to one PostScript unit. If one unit in application space were equal to five PostScript units, then the transformation might look like this:

```
0 792 translate
5 -5 scale
```

Assuming that the coordinate system has already been properly translated and scaled from the PostScript coordinate system to the application coordinate system as above, then the following steps can be used to place the EPS file in the user-chosen box:

1. *left bottom* **translate**

2. ((*right* – *left*)/(*urx* – *llx*)) (*top* – *bottom*)/(*ury* – *lly*) **scale**

3. – (*llx*) – (*lly*) **translate**

where *bottom*, *left*, *top*, and *right* are coordinates of the placement box in application space, and *llx*, *lly*, *urx*, and *ury* are bounding box parameters the EPS file supplies.

As a final example, assume that the PostScript coordinate system has already been transformed to match the application coordinate system, the EPS file bounding box is %%BoundingBox: 20 20 100 100**,** and the user-chosen placement box is the box shown in Figure 5 on page 15. Using the formula and steps above, the transformation before executing the included EPS file would be as follows:

```
20 60 translate
.5 -.5 scale
-20 -20 translate
```

### Set Up a Clipping Path

The importing application should set up a clipping path around the imported EPS file. This can be accomplished by setting a clipping path that corresponds to the bounding box of the imported EPS file after making the PostScript coordinate system transformations or by allowing the user to optionally supply an arbitrary clipping path for special effects.

### Discard the Screen Preview

If an EPS file includes a screen preview in EPSI format, the importing application should discard the preview before sending the document to a printer. Although the EPSI preview is represented by PostScript comments and will not pose a problem when included in the PostScript language file sent to the printer, it takes extra time to transmit the preview.

If the preview in the EPS file is in Macintosh PICT format, do not include the PICT resource in the PostScript language file sent to the printer.

If the preview is in TIFF format or in Microsoft® Windows™ Metafile format, take care to extract the PostScript language code that is to be sent to the printer. See section section 5.2," for details.

If the EPS file does not include a screen preview, the entire EPS file can be included in the PostScript language file sent to the printer.

### Maintain EPSF Version 2.0 Compatibility

The EPSF version 3.0 requires that an EPS file leave the operand and dictionary stacks as they were before the EPSF was executed. However, this was not explicitly stated in earlier versions of the EPSF format. Therefore, before including the EPS file, be sure to count the number of objects on the dictionary and operand stacks. After executing the EPS file, make sure the stacks contain the same number of objects they did before the EPS file was executed.

### Preparation for Including an EPS File

Example 2: shows procedure BeginEPSF, which an application might use to prepare to include an EPS file in its print stream. Execute the BeginEPSF procedure before the EPS file.

**Example 2:**

```
/BeginEPSF { %def
  /b4_Inc_state save def            % Save state for cleanup
  /dict_count countdictstack def    % Count objects on dict stack
  /op_count count 1 sub def         % Count objects on operand stack
  userdict begin                    % Push userdict on dict stack
  /showpage { } def                 % Redefine showpage, { } = null proc
  0 setgray 0 setlinecap            % Prepare graphics state
  1 setlinewidth 0 setlinejoin
  10 setmiterlimit [ ] 0 setdash newpath
  /languagelevel where              % If level not equal to 1 then
  {pop languagelevel                % set strokeadjust and
  1 ne                              % overprint to their defaults.
    {false setstrokeadjust false setoverprint
    } if
  } if
} bind def
```

Example 3: shows procedure EndEPSF, which illustrates how to restore the PostScript state to the way it was before inclusion and execution of the EPS file. Execute the EndEPSF procedure after the EPS file.

**Example 3:**

```
/EndEPSF { %def
  count op_count sub {pop} repeat       % Clean up stacks
  countdictstack dict_count sub {end} repeat
  b4_Inc_state restore
} bind def
```

Example 4: illustrates use of the BeginEPSF and EndEPSF procedures.

**Example 4:**

```
BeginEPSF                               % Prepare for the included EPS file
left bottom translate                   % Place the EPS file
angle rotate
Xscale Yscale scale
-llx -lly translate
...Set up a clipping path...
%%BeginDocument: MyEPSFile
...Included EPS file here...
%%EndDocument
EndEPSF                                 % Restore state, and cleanup stacks
```

## 4 File Types and Naming

EPS files have become a standard format for importing and exporting PostScript language files among applications in a variety of heterogenous environments. This section contains specific information about file types and naming conventions in a variety of environments.

### 4.1 Apple Macintosh File System

The Macintosh file type for application-created PostScript language files is EPSF. Files of type TEXT are also allowed so users can create EPS files with standard text editors. However, the DSC 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 preview image of the EPS file. The file name may follow any naming convention as long as the file type is EPSF. If the file type is TEXT, the extensions .epsf, and .epsi should be used for EPS files with Macintosh-specific and device-independent preview images, respectively. See sections section 5," and section 6."

### 4.2 MS-DOS and PC-DOS File System

The recommended file extension is .EPS. For EPS files that provide an EPSI preview, the recommended extension is .EPI. Because the name and extension may be user-supplied, it is recommended that the application provide a default extension of .EPS or, if the file includes an EPSI preview, the application can provide .EPI as the default extension.

### 4.3 Other File Systems

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

## 5 Device-Specific Screen Preview

The EPS file usually has a graphic screen preview so it can be transformed and displayed on a computer screen to aid in page composition before printing. Depending on the capabilities of the importing application, the user may position, scale, clip, or rotate this screen representation of the EPS file. The composing software should keep track of these transformations and reflect them in the PostScript language code that is ultimately sent to the printer.

The exact format of this screen representation is machine-specific. That is, each computing environment may have its own preferred preview image format, which is typically the appropriate screen representation for that environment. Also, a device-independent screen representation called EPSI is specified in section section 6.” It is recommended that all applications support this format.

### 5.1 Apple Macintosh PICT Resource

A QuickDraw™ representation of the EPS file can be created and stored as a PICT resource in the resource fork of the EPS file. It must 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, the *picframe* and bounding box may not always agree. If there is a discrepancy, the %%BoundingBox: must always be taken as the “truth,” because it accurately describes the area the EPS file will image.

### 5.2 Windows Metafile or TIFF

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 has a binary header added to the beginning that provides a sort of table of contents to the file. This is necessary because there is not a second “fork” in 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.*

The importing application must check the first 4 bytes of the EPS file. If they match the header as shown in Table 1, the binary header should be expected. If the first two match %!, it should be taken to be an ASCII PostScript language file.

**Table 1** *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). If Checksum is FFFF then ignore it. |

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 must follow the guidelines the Windows specification sets forth. It should not set the *viewport* or *mapping mode*, and it should set the *window origin* and *extent*. The application including the EPS file should scale the picture to fit within the %%BoundingBox: comment specified in the EPS file.

## 6 Device-Independent Screen Preview

This screen preview format is designed to allow EPS files to be used as an interchange format among widely varied systems. The preview section of the file is a bitmap represented as ASCII hexadecimal to be simple and easily transportable. This format is called encapsulated PostScript interchange format, or EPSI.

An EPSI file is truly portable and requires no special code for decompressing or otherwise understanding the bitmap portion, other than the ability to understand hexadecimal notation.

The %%BeginPreview: *width height depth lines* and %%EndPreview comments bracket the preview section of an EPSI file. The *width* and *height* fields provide the number of image samples (pixels) for the preview. The *depth* field provides the number of bits of data used to establish one sample pixel of the preview—typical values are 1, 2, 4, 8. An image that 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 an application that does not care may easily skip them. All arguments are integers.

The bit order of the preview image data is the same as the bit order used by the **image** operator. That is, the preview image is considered to exist in its own coordinate system. The rectangular boundary of the preview image has its lower-left corner at (0,0) and its upper-right corner at (*width*, *height*). The byte order is fixed and should be (0,0) through (*width* – 1), then (0,1) through (*width* – 1,1), etc.

## 6.1 Guidelines for EPSI Files

The following guidelines are to clarify a few basic assumptions about the EPSI format, which is intended to be extremely simple because its purpose is for interchange. No system should have to do much work to decipher EPSI files. The format is accordingly kept simple and option free.

- The preview section must appear 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. Arbitrary transfer functions and “flipping” black and white are not supported. Note that in the PostScript language, 0 and 1 have the opposite meaning (0 is black and 1 is white) for the **setgray** operator.

- 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 rather describe the amount of data supplied for the preview. Only the bounding rectangle describes the dimensions.

- 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, because the dimensions of the finished preview are established by the *width*, *height*, and *depth* values.

- All non-hexadecimal characters must 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 **readhexstring** operator.

- Each line of hexadecimal begins with a percent character (%). This makes the entire preview section a PostScript language comment to be ignored by the PostScript interpreter. The file can be printed without modification.

- Although the EPSI hex preview can be sent to the printer, to shorten transmission time it is recommended that the preview image be stripped out of the document before transmitting the file to the printer.

- The data for each scan line of the image must be a multiple of 8 bits long. If necessary, pad the end of the scan line data with 0’s.

Example 5: is a sample EPSI format file. Remember 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) **x** 2. The PostScript language segment simply draws a box, as can be seen in the last few lines.

**Example 5:**

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 80 24
%%Pages: 0
%%Creator: John Smith
%%CreationDate: November 9, 1990
%%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
%%EOF
```

# 7 EPS Example

The following example illustrates the proper use of DSC comments in a typical page description that an application might produce when including an EPS file. For an EPS file that is represented as

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 4 4 608 407
%%Title: (ARTWORK.EPS)
%%CreationDate: (10/17/89) (5:04 PM)
%%EndComments
...PostScript code for illustration..
showpage
%%EOF
```

the including document's page description, including the imported EPS file, would be represented as

```
%!PS-Adobe-3.0
%%BoundingBox: 0 0 612 792
%%Creator: SomeApplication
%%Title: (Smith.Text)
%%CreationDate:  11/9/89 (19:58)
%%Pages: 1
%%DocumentFonts: Times-Roman Times-Italic
%%DocumentNeededFonts: Times-Roman Times-Italic
%%EndComments

%%BeginProlog
/ms {moveto show} bind def
/s /show load def
/SF { %/FontIndex FontSize /FontName SF --
  findfont exch scalefont dup setfont def
} bind def
/sf /setfont load def
/rect { % llx lly w h  % Used to create a clipping path
  4 2 roll moveto
  1 index 0 rlineto
  0 exch rlineto
  neg 0 rlineto
  closepath
} bind def


/BeginEPSF { %def          % Prepare for EPS file
  /b4_Inc_state save def% Save state for cleanup
  /dict_count countdictstack def
  /op_count count 1 sub def % Count objects on op stack
  userdict begin          % Make userdict current dict
  /showpage { } def       % Redefine showpage to be
```

```
null
  0 setgray 0 setlinecap
  1 setlinewidth 0 setlinejoin
  10 setmiterlimit [ ] 0 setdash newpath
  /languagelevel where   % If level not equal to 1 then
    {pop languagelevel     % set strokeadjust and
    1 ne                 % overprint to their defaults
    {false setstrokeadjust false setoverprint
    } if
  } if
}bind def
/EndEPSF { %def
  count op_count sub {pop} repeat
% Clean up dict stack
  countdictstack dict_count sub {end} repeat
  b4_Inc_state restore
} bind def
%%EndProlog

%%BeginSetup
%%IncludeFont: Times-Roman
%%IncludeFont: Times-Italic
%%EndSetup
%%Page: 1 1
%%BeginPageSetup
/pgsave save def
%%EndPageSetup
/F1 40 /Times-Roman  SF
```
*...Set some text with F1...*
```
/F2 40 /Times-Italic SF
```
*...Set some text with F2...*
```
F1 sf
```
*...Set some more text with F1...*
```
F2 sf
```
*...Set some more text with F2...*
```
BeginEPSF
65.2 10 translate           % Position the EPS file
.80 .80 scale               % Scale to desired size
-4 -4 translate             % Move to lower left of the
EPS
4 4 604 403 rect            % Set up clipping path
clip newpath                % Set the clipping path


%%BeginDocument: ARTWORK.EPS
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox:  4 4  608 407
%%Title:  (ARTWORK.EPS)
%%CreationDate:  (10/17/90) (5:04 PM)
%%EndComments
```
*...PostScript code for illustration..*

```
showpage
%%EOF
%%EndDocument

EndEPSF                    % Restore state, cleanup
stacks
pgsave restore
showpage
%%EOF
```

# Appendix: Changes Since Earlier Versions

This content of this document is exactly the same as the specification in Appendix H of the *PostScript Language Reference Manual, Second Edition*.

## Changes Since Version 2.0

Detailed DSC comment descriptions have been left out of this specification. When developing an application that will support EPS files, the DSC version 3.0 (see the *PostScript Document Structuring Conventions Specifications* available from the Adobe Systems Developers' Association) should be used with this specification.

The following conditionally required DSC comments were added to this specification as of version 3.0:

%%Extensions:
%%LanguageLevel:
%%DocumentNeededResources:
%%IncludeResource:
%%Begin(End)Document:

### Changes Relevant to Applications Producing EPS Files

To help avoid ambiguities, section 2, "Guidelines for Creating EPS Files," has been added. This new section has several guidelines for producing EPS files. Following these guidelines will help ensure that an EPS file can be reliably included in documents without causing any annoying side effects. Also, these new rules allow applications to easily determine if an EPS file is compatible with version 3.0 of the EPS file format. The following is an overview of the new guidelines:

- %%Begin(End)Preview: comments must bracket an EPSI preview.
- There is a list of illegal operators that must not be used in an EPS file.

- There is a list of restricted operators. If these operators are used in an EPS file, they must be used in accordance with the guidelines presented in Appendix I of the *PostScript Language Reference Manual, Second Edition*.

- The operand and dictionary stacks must be returned to the state that they were in before the EPS file was executed.

- It is strongly recommended that an EPS file make its definitions in its own dictionary or dictionaries.

- An EPS file must not rely on procedures defined outside of the server loop, such as procedures defined in the LaserPrep file.

### Changes Relevant to Applications Importing EPS Files

To help clarify the responsibilities of an application including an EPS file, section 3, "Guidelines for Importing EPS Files," specifies the following new rules:

- The including application must define **showpage** as null.

- The application must prepare the graphics state for the EPS file.

- The application must give the EPS file a clear operand stack.

- The application must surround the included EPS file by the %%Begin(End)Document: comments.

# Index


## A

Apple Macintosh file system
  EPS files and 18
Apple Macintosh PICT resource
  EPS files and 19
applications
  EPS files and 2–3, 27–28

## B

%%BeginDocument:
  EPS files and 12
%%BeginPreview:
  EPS files and 20
%%BoundingBox:
  EPS files and 14

## C

changes
  EPSF format 27–28
**clear**
  EPS files and 7, 11
**cleardictstack**
  EPS files and 7
clipping path
  EPS files and 16
comment(s)
  conditionally required for EPS files 5–??
  recommended for EPS files 6
  required for EPS files 3–5
compatibility
  EPS files and 16
conditionally required comments
  EPS files and 5–??
coordinate system transformation
  EPS files and 12–16

## D

device-independent screen preview
  EPS files and 20–22
device-specific screen preview
  EPS files and 18–20
dictionar(ies)
  EPS files and 6
displaying EPS files 9–10
DOS file system
  EPS files and 18, 19

## E

%%EndDocument
  EPS files and 12
%%EndPreview
  EPS files and 20
.EPI file extension 18
.EPS file extension 18
.epsf file extension 18
.epsi file extension 18
EPS (encapsulated PostScript) files
  creating 3–9, 27–28
  device-independent screen preview and 20–22
  device-specific screen preview and 18–20
  displaying 9–10
  example 23–25
  file types and 18
  illegal operators 6
  importing 9–??, 28
  naming 18
  preparation for including 16
  restricted operators 6
EPSF (encapsulated PostScript file) format 1–28
  background 1–3

changes to 27–28
EPSI (encapsulated PostScript interchange) files
guidelines for 21–22

## G

graphics state
EPS files and 7, 11

## I

**image**
EPSI files and 20
importing EPS files 9–??, 28
initializing variables
EPS files and 7–8

## M

Macintosh file system
EPS files and 18
Metafile (Windows)
EPS files and 19

## N

naming conventions
EPS file 18

## O

operand stack
EPS files and 11

## P

PICT resource
EPS files and 19
portability of EPS files 8
preview
screen 18–22

## R

recommended comments
EPS files and 6
required comments
EPS files and 3–5
**restore**
EPS files and 10

## S

**save**
EPSF files and 10
screen preview(s)
device-independent 20–22
device-specific 18–20
EPS files and 16
**showpage**
EPS files and 10, 28
stack(s)
EPS files and 6, 11

## T

tag image file format (TIFF)
EPS files and 19
transformation(s)
coordinate system 12–16

## V

variables
initializing 7–8

## W

Windows Metafile
EPS files and 19