POSTSCRIPT™
Software From Adobe

Adobe Illustrator File Format Specification

Adobe Developer Support

Version 3.0

Draft
28 October 1992

DRAFT

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 LPS5007

Contents

Adobe Illustrator File Format Specification

1 Why This is a Draft Specification

This specification is still in draft form. We recognize that this specification is not yet as definitive as it should be, nor is it a clear, concise explantation of the Adobe Illustrator file format. However, we do feel that this document is suitable for those willing to use it as a reference while simultaneously examining real Adobe Illustrator files. In cases where the documentation and the sample file are in disagreement, the actual file should be used as the "correct" interpretation.

2 Introduction

This Technical Note describes the format ofAdobe Illustrator 3.x documents (files). An Adobe Illustrator document is a PostScript® language page description that conforms to the *Adobe Document Structuring Conventions Specification*, version 3.0. Any PostScript language interpreter can execute the page description to render the illustration on a display or the printed page. Page composition systems and other illustration-editing applications can also import such a document. Programs such as spoolers may parse page descriptions to extract information about resources that the document requires—fonts or procedure sets, for example.

Adobe Illustrator can store a document in *encapsulated PostScript file* (EPS) format; however, its default format is not an official EPS format. Other formats, explained in section section 14," include the EPS file format. An EPS file can include an image of the illustration suitable for previewing on specific computer platforms, and a user can select whether the preview image is in color or black and white. A page composition system can use a preview image for picture placement, scaling, cropping, and previewing.

Although an intimate knowledge of the PostScript language is not essential for understanding this document, you should be familiar with all terms used in the *Adobe Illustrator User Guide,* and several additional concepts:

- Adobe Illustrator descriptions are based on the path, as described in *PostScript Language Reference Manual, Second Edition*. Adobe Illustrator paths, however, are not as general as paths created for execution by the PostScript language.

- You can put opaque ink on the page either by *stroking* the path to trace it with a line of a given thickness (as with a pen), or by *filling* the path to put ink everywhere inside the path.

- The thickness of the line used to stroke the path, the color of the ink used, and so forth, are attributes of the PostScript *graphics state*. A PostScript language program can change the graphics state and thus control how a path is marked. In Adobe Illustrator, fill and stroke attributes maintain their own states. Whenever the graphic state attributes that would affect filling or stroking are changed, Illustrator appropriately modifies procedures that carry out the fill or stroke.

- Text is a special case of the more general drawing facilities of the PostScript language. Pre-defined PostScript programs (called *fonts*) render individual characters by constructing and then filling or stroking a path.

- Coordinate values written out for objects in the Adobe Illustrator file are offsets from the ruler origins. Generally, the origin is in the lower left-hand corner of the space, the *x*-axis extends horizontally to the right, and the *y*-axis extends vertically upward. However, changes in the ruler origin can modify this. The length of a unit along each axis is $^1/_{72}$ inch, which approximates a printer's point ($^1/_{72.27}$ inch).

For additional information about the PostScript language, see the *PostScript Language Reference Manual, Second Edition*, Addison-Wesley, ISBN 0-201-18127-4. A full discussion of Document Structuring Conventions and Encapsulated PostScript Files also appear as appendices in that document.

3 Overview

Adobe Illustrator creates documents (files) that conform to Adobe Systems' *document structuring conventions*. A PostScript language document description that minimally conforms to the Document Structuring Conventions has two main parts: a *prolog* and a *script*. The prolog portion encapsulates information needed by other programs to interpret the file, such as the bounding box that contains all marks on the page. It also contains lists of resources, such as fonts, that the pages may need, and PostScript language procedure definitions and the variables that are used in the document's page descriptions. Documents that conform to the *Document Structuring Conventions* do not execute any PostScript code in the prolog (other than definitions) and should not define new procedures or global variables in the script (although they may well modify the behavior of procedures defined in the prolog or set the value of global variables).

Adobe Illustrator uses a subset of the full PostScript language to describe the graphic elements on a page. The subset is purely declarative; properly setting the position or attributes of graphic elements on the page does not require complex computation in the PostScript language. Pages expressed this way print or image faster, and generally are smaller in size. The language subset is defined in *Resources* (sometimes called *Procedure Sets* or *proc sets*), logically grouped together in the prolog with explicit methods for their initialization and termination.

Note *Don't confuse the term "resources" as used in Adobe Illustrator files with a file's resource fork as used in the Macintosh environment, resources as defined in PostScript Language Reference Manual, Second Edition, or procedure sets encapsulated as modules of PostScript language code.*

After the prolog, the main body of the document is the *script*. A script has three logical sections: a *Setup* sequence that initializes and activates the Resources defined in the prolog, a sequence of *descriptive operators*, and a *Trailer* that deactivates Resources. The script holds the operators, which are sequences of graphic elements and which are written in the language subset as defined in the prolog. These sequences consist of collections of data elements, graphic attribute definitions, and calls to the procedures defined in the prolog's Resources.

The prolog part of a document contains information needed by other programs that may interpret the file. Illustrator includes the different parts of the prolog as needed by the file. For example, the prolog contains the PostScript language definition of procedures and variables used in the individual page descriptions. It can contain a set of comments collectively referred to as a "header" that can provide certain information to Illustrator and other programs that read and interpret those comments. The prolog also defines the collection of PostScript procedures that implement the operators of the special language subset. Adobe Illustrator 3.x places no prolog in its default "no header" save format; it places a header in its other save formats. The main body of the file is the script which describes the document by using the illustration language.

Generally speaking, Illustrator works in the following way: graphic state attributes are set up, rendering routines prepared, the path is expressed, and then the generic procedures are called the carry out the fill, stroke, or other rendering.

3.1 Comments

Stylized comments form the document structuring information in a PostScript program.The PostScript langugage treats as a comment any line where the first non-whitespace character is **%**. Apart from the first comment in a file, the structuring comments all have the form

```
%%Keyword{: arguments}
   or
   %AI3_Keyword{: arguments}
```

Many structuring comments require information in addition to the keyword. This information is separated from the keyword by a colon and continues on to the newline character that terminates the comment.

3.2 Syntax

The Adobe Illustrator document format can be described using BNF (Backus-Naur form) notation:

```
<xyz>::=            abc <def> ghi |
                    <k> j
```

A token enclosed in angle brackets names a class of document component, while plain text appears verbatim or with some obvious substitution. The grammar rules have two parts. On the left of the ::= definition symbol is the name of a class of component. In the example above, the class is *xyz*. On the right of the definition symbol is a set of one or more alternative forms that an *xyz* component might take in the document. The alternative forms are separated by the vertical bar character (|).

Each line on the right-hand side of the definition corresponds to one line in the document. If the only content on the right-hand side is another class name, then the line may represent more than one line in the document. Single letter components, such as <*A*>, refer to the corresponding illustration language operator *A*. The notation *{…}* means that the items enclosed in braces are optional. If an asterisk follows the braces, the objects inside the braces may be repeated *zero* or more times. The notation <…>+ means that the items enclosed within the brackets may be repeated *one* or more times.

For example, using BNF notation, you can described the overall structure of an Adobe Illustrator document as

```
<document> ::=      <prolog>
                    <script>

<prolog> ::=        %!PS-Adobe-3.0 EPSF-3.0
                    <header comments>
                    %%EndComments
                    {<proc set>}*
                    %%EndProlog

<script> ::=        <setup>
                    {<object>}*
                    {<page trailer>}
                    <document trailer>
                    %%EOF
```

The full document syntax structure appears in section section 17.” The syntax and semantics of the individual operators of the illustration language are defined in later sections of this document. Each operator definition follows this form:

operand$_1$…operand$_m$ **op –**

The functionality of the **op** operator follows here.

This notation means that the operator **op** takes operands from *1* through *m* and performs some operation. Each operand is characterized either by its data type (for example, *integer*) or a more meaningful name, such as *linewidth*. In the latter case, the range of legitimate values appears in the description. A dash (–) on the left of the operator indicates that an operator requires no operands; a dash on the right indicates that the operator leaves nothing on the stack.

3.3 Differences Between Versions

There are several versions of Adobe Illustrator available. These include *Adobe Illustrator 3.x, Adobe Illustrator for Windows Version 4.x,* and *Adobe Illustrator Japanese Edition*. This document describes the file format for Adobe Illustrator 3.x; for most purposes, the format for Adobe Illustrator for Windows Version 4.x is the same as that for 3.x, with differences noted in section section 15.” Adobe Illustrator 88 and Adobe Illustrator Japanese Edition are described in the Adobe Technical Note **XXX-XXX**.

Where differences bear on which operands to use for particular operators or which comments to include in a header, those differences are indicated in the descriptions of the individual components of the document. Parts of the document or individual operators that are used exclusively by one or another versions are so marked. Generally, Adobe Illustrator software ignores comments and operators that are meant for other versions.

This document uses the following version abbreviations:

Adobe Illustrator, version 3.xAI3
Adobe Illustrator for Windows Version 4.xAIWin
Adobe Illustrator NeXT, version 3.xAINeXT
Adobe Illustrator Japanese EditionAIJE
Adobe Illustrator 88AI88

4 Prolog

The syntax for an Adobe Illustrator 3.x document prolog is:

<prolog> ::=	%!PS-Adobe-3.0 EPSF-3.0
	<header comments>
	%%EndComments
	{<proc set>}*
	%%EndProlog

%!PS-Adobe-3.0 EPSF-3.0

The first line of the file is a unique comment that identifies the version of the Document Structuring Conventions to which the document conforms. In this case, the file conforms to version 3.0. The first line may also specify that the file conforms to a version of the Encapsulated PostScript File format (EPSF). Both numbers must correspond to the specific versions used in writing out the file.

%%BeginProlog

The **%%BeginProlog** comment marks the beginning of the prolog section.

%%EndProlog

The **%%EndProlog** comment marks the end of the prolog section.

4.1 Header

The header for the prolog body of an Adobe Illustrator document follows the version-identifying first line of the file. The syntax for the prolog header is

<header> ::= <header comments>
%%EndComments

%%EndComments

The **%%EndComments** comment marks the end of the header part of the prolog.

The sequence of header comments is a subset of those listed in Figure 1. The syntax for each comment is described informally. Almost all header comments are optional. Comments required by Adobe Illustrator in all documents are marked **[Required]**. Some comments are required only if a specific feature is used in an illustration. Such comments are marked **[As Necessary]**. Those comments that are specific to Adobe Illustrator version 3.x have the form

```
%AI3_Keyword{: arguments}
```

Figure 1 . *List of typical Adobe Illustrator header comments; comments can vary from file to file*

```
%%Creator: Adobe Illustrator(TM) 3.0.1
  %%For: (username) (organization)
  %%Title: (illustration title)
  %%CreationDate: (date) (time)
  %%BoundingBox: llx lly urx ury
  %%DocumentProcessColors:  keyword
  %%DocumentFonts:  font…
  %%+font…
  %%DocumentFiles:  filename
  %%+filename…
  %%DocumentSuppliedResources:  procset Adobe_packedarray ver-
  sion revision
  %%+ procset Adobe_cmykcolor version revision
  %%+ procset Adobe_cshow version revision
  %%+ procset Adobe_customcolor version revision
  %%+ procset Adobe_pattern_AI3 version revision
  %%+ procset Adobe_typography_AI3 version revision
  %%+ procset Adobe_IllustratorA_AI3 version revision
  %AI3_ColorUsage:  keyword
  %AI3_TemplateBox: llx lly urx ury
  %AI3_TemplateFile:  pathname
  %AI3_TileBox:  llx lly urx ury
  %AI3_DocumentPreview:  previewtype
```

The individual lines in the header are specified as follows.

%%Creator: Adobe Illustrator(TM) *version*

The **%%Creator** comment identifies the application that generated the PostScript language document. The *version* number is arbitrary text terminated by a newline character.

%%For: (*username*) (*organization*)

The **%%For** comment identifies the user who created the file and the organization to which the user belongs. Both *username* and *organization* are valid PostScript language strings. The PostScript language string escape sequences for including characters outside the printable ASCII character set and for representing characters such as *(* and *)* and other special characters are discussed in *PostScript Language Reference Manual, Second Edition*.

%%Title: (*illustration title*)

The **%%Title** comment provides an arbitrary text title for the document. The title is a valid PostScript language string.

%%CreationDate: (*date*) (*time*)

The **%%CreationDate** comment gives the date and time that the document was created. The variables *date* and *time* are valid PostScript language strings.

%%BoundingBox: *llx lly urx ury*

[Required]. The **%%BoundingBox** comment specifies the imaginary box that encloses all marks painted on the page. Specify the integer coordinates in the default user coordinate system. Negative numbers are allowed.

%%DocumentProcessColors: *keyword*

The **%%DocumentProcessColors** comment specifies which of the process colors identified by the keywords **Cyan**, **Magenta**, **Yellow**, and **Black** the document uses. This comment is used primarily by programs producing color separations.

%%DocumentCustomColors: (*customcolorname*)
%%+ (*customcolorname*)

[As Necessary]. The **%%DocumentCustomColors** comment enumerates the names of the custom colors used in the document. The names are valid PostScript language strings enclosed in parentheses. For example, the PANTONE® colors are identified by names such as **(PANTONE 156 CV)**. You may continue the list of custom color names on subsequent lines, each of which must begin with the **%%+** prefix.

%%CMYKCustomColor: *cyan magenta yellow black (customcolorname)*
%%+ *cyan magenta yellow black (customcolorname)*

[As Necessary]. The **%%CMYKCustomColor** comment specifies an approximation for the named custom color in terms of the four components of process color: *cyan*, *magenta*, *yellow*, and *black*. Each component value must be a real number in the range 0.0 to 1.0. The component values are analogous to the arguments to the PostScript language operator **setcmykcolor**.

%%DocumentFonts: *font…*
%%+*font…*

[As Necessary]. The **%%DocumentFonts** comment enumerates the names of the PostScript language font programs that the document uses. Fonts listed in the **%%DocumentFonts** comment are also included in any files which are themselves included (placed) within an Adobe Illustrator document. Omit this comment if the document uses no fonts.

You may need to download a font to the PostScript device before it can properly execute a document description.

%%DocumentFiles: *filename*
%%+*filename…*

[As Necessary]. The **%%DocumentFiles** comment names the files that a program must import to render the illustration. Another comment (**%%IncludeFile**) marks the site within the illustration at which the file is needed. Omit this comment if no files are to be imported into the document. See section 9” for more information on including files.

%%DocumentSuppliedResources: procset Adobe_packedarray *version revision*
%%+ procset Adobe_cmykcolor *version revision*
%%+ procset Adobe_cshow *version revision*
%%+ procset Adobe_customcolor *name version revision*
%%+ procset Adobe_pattern_AI3 *version revision*
%%+ procset Adobe_typography_AI3 *version revision*
%%+ procset Adobe_IllustratorA_AI3 *version revision*

[AI3]. These comments (the above list is only an example) indicate that the named procedure sets and resources are both required *and* defined by the document. The **%%+ procset** construction indicates a continuation of the **%%DocumentSuppliedResources** comment.

Comments appear only for the actual procedure sets needed by the illustration; they are not present when the file is saved in its "no-header" form.

%%DocumentNeededResources

[AI3] This comment tells what procsets are required by the document that are not included within it.

%%IncludeResource

[AI3] This comment lists a specific resource to include in a document. It is present only when the file is saved in its "no-header" format.

%AI3_ColorUsage: *keyword*

The **%AI3_ColorUsage** comment indicates whether the document uses only black or colored ink, indicated by the keyword **Black&White** or **Color**, respectively.

[AIWin, AI88, AIJE]. This comment is named **%%ColorUsage**.

%AI3_TemplateBox: *llx lly urx ury*

The **%AI3_TemplateBox** comment specifies the bounding box that encloses all samples in the document's template. For more information on templates, see *Adobe Illustrator User Guide.* Specify the coordinates as integers or reals in the default user coordinate system. Each sample in the template is assumed to be $^1\!/_{72}$ inch square. The width *(urx–llx)* and height *(ury–lly)* of the template box must be integers. If a document has no template, the width and the height of the template box must be zero.

When Adobe Illustrator opens a document, it centers the coordinate $((llx + urx)/2, (lly + ury)/2)$ in the drawing area.

[AIWin, AI88, AIJE]. This comment is named **%%TemplateBox**.

%AI3_TemplateFile: *pathname*

[As necessary]. The **%AI3_TemplateFile** comment specifies the template for the illustration. If no name is given, no template is used. The format for specifying the template is:

<volume name>::<directory id (a number)>:<filename>

%AI3_TileBox: *llx lly urx ury*

The **%AI3_TileBox** comment is used only on the Macintosh version of Adobe Illustrator. It specifies the bounding box of the imageable area of the current page size. See *Adobe Illustrator User Guide* for more information about tiles and the drawing area. The initial ruler position is centered in this box.

[AIWin, AI88, AIJE]. This comment is named **%%TileBox**.

4.2 Artwork and Ruler Origin

All artwork elements, as well as the Bounding Box, Template Box, and Tile Box, are written out in coordinates relative to the *ruler origin*, with *y* increasing up and *x* increasing to the right, and bounds in the order left, bottom, right, top.

The template, if there is one, is always centered on the artboard. If there is no template associated with the artwork, the **%AI3_TemplateBox** comment describes a degenerate box positioned at the center of the artboard. Since it is written out in ruler-relative coordinates, the center of the template bounding box can be used to establish the ruler origin by measuring backwards from the center of the current artboard (that is by measuring *x* to the left of the center of the template bounding box and *y* down from the center). It is done this way since the size of the artboard may change between the version of Illustrator a file is saved with and the version it is read back in under. In such a situation, it is the *centers* of the two artboards that need to be aligned.

That is, when the file is opened, the Template Box rectangle is read in, and then the ruler origin is calculated as:

$$x = (\text{artboard width} - \text{templateBox.left} - \text{templateBox.right})/2$$

$$y = (\text{artboard height} + \text{templateBox.top} + \text{templateBox.bottom})/2$$

(This *x,y* is in Macintosh space, where *y* increases down, unlike the Illustrator file format, where y increases up.)

The position of the ruler, of course, is only really meaningful inside Illustrator or another illustration editing program that wishes to import Illustrator files and keep the ruler position intact. For applications that do not care about Illustrator's ruler position, it is sufficient to choose as an origin any point pertinent to the importing application, such as one of the corners of the bounding box, and apply to all the points in the artwork the translation that would take that point to 0,0.

5 Script Setup

The syntax for the script setup section of an Adobe Illustrator 3.x document is

```
<script> ::=              <setup>
                          {<object>}*
                          {<page trailer>}
                          <document trailer>
                          %%EOF

<setup> ::=               %%BeginSetup
                          {%%IncludeFont: font}*
                          {<proc set init>}*
                          <font encoding>
                          <pattern defs>
                          %%EndSetup

<page trailer> ::=        %%PageTrailer
                          gsave annotatepage grestore showpage

<document trailer> ::=    %%Trailer
                          {<proc set termination>}*
```

The following sections describe the individual components of the setup section of an Illustrator document.

5.1 Specifying Particular Fonts

The comment **%%IncludeFont** specifies a font that appears in he document. Adobe Illustrator 3.x checks to see whether that font is available and uses it if it is. If the font is not available, Adobe Illustrator uses another font.

5.2 Initializing Resources

Adobe Illustrator customarily *initializes* those resources (proc sets) required by the document. A corresponding *termination* appears in the document trailer.

5.3 Fonts and Encodings

The mapping between ASCII characters and glyphs in a font is different from the standard mapping used in a PostScript font. Therefore, to print a document correctly, the mapping must be changed for each PostScript font used in an Adobe Illustrator document. The action of altering the mapping between character codes and glyphs is called *re-encoding* the font.

The syntax for re-encoding a font in an Adobe Illustrator document is

```
<font encoding> ::=     [
                        {<re-encoding pairs>}*
                        <Te>
                        {<re-encoding>}*

<re-encoding> ::=       %AI3_BeginEncoding
                        newFontName oldFontName
                        <TZ>
                        %AI3_EndEncoding <font type>

<font type> ::=         AdobeType|TrueType
```

The **TE** operator sets the standard encoding for the platform on which the Illustrator file is being executed. The **TZ** operator performs the re-encoding. Once encoding has been specified, the **Tf** operator can specify the font name and the font size.

[<encoding pairs> **TE**

The **TE** operator sets the standard platform font encoding. Note that there is no right bracket following the parameter.

DRAFT

[newFontname oldFontName direction fontScript useDefault **TZ**
[encodingPairs newFontName oldFontName direction fontScript useDefault **TZ**
[newFontName oldFontName direction fontScript useDefault [w0 w1…wn] **TZ**
[encodingPairs newFontName oldFontName direction fontScript useDefault [w0 w1…wn] **TZ**

The **TZ** operator creates a new font from an existing font by changing portions of the new font's encoding vector. The first two forms are for Type 1 font programs; the second two forms are for Multiple Master typefaces. The forms with the *encodingPairs* operand are used when changing font encoding. These encodings are platform-specific to the Macintosh computer; the NeXT computer, for example, may not regularly need encodings because it uses the Display PostScript System.

The *encodingPairs* operand is a list of encoding numbers and literal glyph names organized as follows:

$\mathtt{code}_1$ $\mathtt{/name}_{11}$ $\mathtt{/name}_{12}$ … $\mathtt{/name}_{1j}$
$\mathtt{code}_2$ $\mathtt{/name}_{21}$ $\mathtt{/name}_{22}$ … $\mathtt{/name}_{2k}$
…
$\mathtt{code}_n$ $\mathtt{/name}_{n1}$ $\mathtt{/name}_{n2}$ … $\mathtt{/name}_{nl}$

where each *code* is in the range 0 to 255 and each *name* is the literal glyph name. The / preceding each name is the syntax used to distinguish a PostScript language literal name from an executable name. This list describes a set of sequences of glyph names to install in the new encoding vector. Each sequence begins with the character index of the first name to be replaced. Subsequent names are replaced up to the next character index entry in *encodingPairs*, at which point a new sequence of replacement names begins, starting with the one at the new character index.

The *newFontName* and *oldFontName* operands are the PostScript names for the new font and the original font. These names must be the same as the names given in the **%%BeginEncoding** comment. The *direction* operand is zero.

Note *For versions of Adobe Illustrator other than Adobe Illustrator 3.x and Adobe Illustrator Japanese Edition, there is no* direction *operand.*

For composite fonts (such as Japanses language fonts) the *direction* operand is 0 for horizontal writing and 1 for vertical writing. The *fontScript* operand is 0 for Roman typefaces and 1 for composite typefaces. For composite fonts the [*encodingPairs* list must have a single left bracket.

The *defaultEncoding* operand controls whether the **TE** encoding is used (1) or not (0). If the font is a Multiple Master typeface, the final array operand is the weightVector of the Multiple Master instance.

Figure 2 shows how to use the **TZ** operator. The example derives a new font named *_Times-Roman* from the original *Times-Roman* font. It replaces three sequences of characters within the encoding vector; the first one-character sequence is number 39, the second one-character sequence is number 96, and the third sequence replaces the characters numbered 128 and above.

Figure 2 . *Re-encoding Times Roman with the TZ operator*

```
%%BeginEncoding: _Times-Roman Times-Roman
   [
   /_Times-Roman/Times-Roman 0 0 1 TZ
   %%EndEncoding
```

5.4 Pattern Definition

The script setup section of a document defines the patterns used by Adobe Illustrator. A pattern is essentially just another Adobe Illustrator illustration that can be drawn repeatedly. You cannot use placed files nor graph objects within a pattern, but patterns can include paths and text. Therefore, parts of the description of how patterns are defined necessarily refers to the description of how an illustration is described in the document script section.

Each pattern is defined by a rectangle used to *tile* the drawing area. The illustration within that rectangle constitutes the pattern used when a path is stroked or filled with a pattern.

The syntax for a pattern is

<pattern defs> ::=	{<pattern>}*
<pattern> ::=	%AI3_BeginPattern: (*patternname*) <E> %AI3_EndPattern

(patternname) llx lly urx ury [<layer list>] **E**

The **E** operator defines a new pattern called *patternname* using the *layer list* for which the bounding box is specified by (*llx, lly*) and (*urx, ury*).

Section section 6” describes how a general illustration is composed of layers, each of which is drawn on top of lower layers which appear earlier in the sequence. Because a pattern is really just a mini-illustration, it too is composed of layers. The syntax for the list of layers is

<layer list> : :=	{<layers>}*
<layer> ::=	<@> <&>

Each layer of the pattern consists of two parts. The first part defines the color to be used for filling and stroking the pattern. The second part defines the other style parameters and the paths for drawing the pattern.

(colordefinition) **@**

The **@** operator defines the color and overprinting style for the associated layer in the pattern. The *colordefinition* parameter begins with a specification of the overprinting option. For more information on overprinting, see the definitions of operators **O** and **R** in section section 6.3.” The filling or stroking color is then defined using the simple gray operators (**g** and **G**), the process color operators (**k** and **K**), or the custom color operators (**x** and **X**). All color operators are defined in section section 6.3.”

(tiledefinition) **&**

The **&** operator defines the tile for the pattern layer that includes the drawing styles and illustration components. This is identical to the representation of objects in the document script except that the color components and both parts of the object are specified separately as PostScript language strings, which are enclosed in parentheses. The use of strings limits the size of a pattern layer definition to 64K bytes.

Whenever a pattern background is filled or stroked, the first layer of the pattern defines the background for the tile. If the pattern tile rectangle is filled, then you must first use the special **_** (underbar) operator to specify a fill. If the pattern tile rectangle is stroked, then normal path construction of the rectangle specifies the pattern tile to stroke. Breaking down the filling and stroking of the pattern tile results in performance optimization when imaging.

_

The _ (underbar) operator signals the pattern machinery that the tile rectangle for the path is to be filled with the fill color previously specified to the @ operator. Figure 3 shows an example pattern definition.

Figure 3 . *An example pattern definition*

```
%%BeginPattern:  (no vegetation)
   (no vegetation) 105 561.875 138 594.875 [
   (0 O 0 R 0.03 0,05 0.15 0 (PANTONE 468 CV) 0 x 0.03 0.05 0.15 0
   (PANTONE 468 CV) 0 X) @
   _ &
   (0 O 0 R 0.125 0.25 0.525 0 (PANTONE 465 CV) 0 x 0.125 0.25
   0.525 0 (PANTONE 465 CV) 0 X) @
   (
   0 i 0 J 0 j 1 w 4 M []0 d
   %%Note:
   105 561.875 m
   105 594.875 L
   138 594.875 L
   138 561.875 L
   105 561.875 L
   s
   ) &
   (0 O 0 R 0 0.15 0.4 0 (PANTONE 156 CV) 0 x 0 0.15 0.4 0 (PANTONE
   156 CV) 0 X) @
   (
   0 i 0 J 0 j 0.3 w 4 M []0 d
   %%Note:
   105 599 m
   105 560 I
   S
   …
   138 599 m
   138 560 I
   S
   ) &
   ] E
   %%EndPattern
```

The example in Figure 3 defines a pattern called “no vegetation” where the pattern tile is both filled and stroked.

First is the pattern name, the bounding box for the pattern tile, then the layer list. The first item in the layer list specifies the custom color *PANTONE 468 CV* as the fill color of the pattern tile. The _ operator specifies a fill of the pattern tile. The next layer in the pattern specifies a stroke of the pattern tile. The custom color *PANTONE 465 CV* is the stroke color. Following the color specification is the drawing of the pattern tile itself. See section section 6.2” for a description of the style options selected at the beginning of the tile definition. Each path in the tile layer is then specified with a sequence of **m** (**moveto**) and **l** (**lineto**) operations. Each path in the layer is stroked by the **S** operator.

DRAFT

Once the pattern tile is drawn and stroked, the elements of the pattern follow; in this example, they are vertical lines which are all stroked with the same color.

6 Script Body

An illustration is composed of a sequence of graphic elements. The PostScript imaging model is based on opaque ink so that elements later in the sequence are effectively “on top of” other elements earlier in the sequence. Thus, later elements can obscure earlier elements.

Fill and stroke attributes are *state-based*; that is, once set, they remain set until changed.

The syntax for the sequence of elements is

```
<object> ::=            {<A>}(object locking)
                        <path object>|
                        <path mask>|
                        <composite object>|
                        <text object>|
                        <placed art object>|
                        <subscriber object>|
                        <graph object>|
                        <PostScript document>

<path object> ::=       <paint style>
                        <path geometry>
                        <path render>|<*> (<*> indicates
                        guide operator)

<path mask> ::=         <paint style>
                        <path geometry>
                        <h>|<H>
                        <W>
                        <path render>

<composite object> ::=
                        <group object>|
                        <group with a mask>|
                        <compound path>|
                        <compound path mask>|
                        <wraparound group>

<group object> ::=      <u>
                        <object>+
                        <U>
```

```
<group with a mask> ::=
                        <q>
                        {<object>}*
                        {<masked object>}*
                        <Q>

<masked object> ::=     <mask>|<object>

<mask> ::=              <path mask>|<compound path mask>

<compound path> ::=     <*u>
                        <compound path element>+
                        <*U>

<compound path element> ::=
                        <path object>|<compound group>

<compound group> ::=
                        <u>
                        <compound path element>+
                        <U>

<compound path mask> ::=
                        <*u>
                        <compound path mask element>+
                        <*U>

<compound path mask element> ::=
                        <path mask>|<compound mask group>

<compound mask group> ::=
                        <compound mask bottom group>|
                        <compound mask non-bottom group>

<compound mask bottom group> ::=
                        {<A>}
                        <q>
                        <path mask>+
                        <Q>

<compound mask non-bottom group> ::=
                        {<A>}
                        <u>
                        <compound mask group>+
                        <U>
```

The following sections explain the individual operators for describing graphic objects.

6.1 Locked Object Operator

flag **A**

The **A** operator specifies whether the object defined immediately after the operator can be selected when editing the document with Adobe Illustrator. The *flag* operand may be either 0 or 1. If *flag* is 0, the object may be selected for editing. If *flag* is 1, the object is “locked” and may not be selected. This state remains in force for every subsequent element unless specifically changed.

6.2 Graphics State Operators

[array] phase **d**

The **d** operator is equivalent to the PostScript language **setdash** operator. It sets the dash pattern parameter in the graphics state, to control the dash pattern of subsequently stroked paths. The *array* of values specifies distances in user space for the length of dashes and gaps, respectively, in the dash pattern. The *phase* operand determines the phase of the dash pattern with respect to the start of the path. It is specified as a distance in user space into the pattern at which to begin marking the path. The initial dash pattern is a solid line.

Note *Adobe Illustrator does not provide an interface for users to adjust this phase parameter. Adobe Illustrator preserves this phase for documents that the user edits and saves.*

flatness **i**

The **i** operator is equivalent to the PostScript language **setflat** operator, which sets the flatness parameter in the graphics state. The *flatness* parameter specifies the accuracy or smoothness with which curves are rendered as a sequence of flat line segments. Specifically, it sets the maximum permitted distance in device pixels between the mathematical path and a given straight line segment. The default value for the *flatness* parameter is 0.0. If *flatness* is specified as 0, *flatness* is set by Adobe Illustrator to the *flatness* parameter in effect when the prolog was executed; in most cases, that is the device’s default flatness. This may be the device’s default flatness, it may be a value you have entered, or it may be a value inherited from an encapsulating context. Acceptable range is 0 to 100.

flag **D**

The **D** operator determines the *winding order* of the object. The PostScript language fills areas to the left of the path direction. The operand *flag* is 0 for a clockwise path direction and 1 for a counter-clockwise direction.

linejoin **j**

The **j** operator is equivalent to the PostScript language **setlinejoin** operator, which sets the line join parameter in the graphics state. This parameter specifies the shape to put at corners in paths when they are stroked. The *linejoin* parameter may be 0 for mitered joins, 1 for round joins, and 2 for beveled joins. The initial *linejoin* is 0.

linecap **J**

The **J** operator is equivalent to the PostScript language **setlinecap** operator, which sets the line cap parameter in the graphics state. If *linecap* is 0, Illustrator uses butt end caps and squares off line ends. If *linecap* is 1, it uses round end caps. If *linecap* is 2, it uses projecting square end caps. The projection extends beyond the end of the line by a distance which is half the line width. The initial *linecap* value is 0.

miterlimit **M**

The **M** operator is equivalent to the PostScript language **setmiterlimit** operator, which sets the miter limit parameter in the graphics state. The *miterlimit* operand must be a real number greater than one. When you have specified mitered joins and two line segments meet at a sharp angle, it is possible for the miter to extend far beyond the thickness of the line stroking the path. The miter limit imposes a limit on the ratio of the length of the miter to the line width. When the limit is exceeded, the file prints with a bevel join instead of a miter. The initial miter limit is 4.

linewidth **w**

The **w** operator is equivalent to the PostScript language **setlinewidth** operator, which sets the line width parameter in the graphics state. This parameter controls the thickness of the line used to stroke a path and is specified as a distance in user space. The initial *linewidth* is 1.0. A line width of 0 is acceptable; when Illustrator prints the file, this is interpreted as the thinnest line width that can be rendered at device resolution (not recommended on high-resolution devices because the line may be nearly invisible).

6.3 Color Operators

The settings for color operators and gray scale can extend to four decimal places.

gray **g**

The **g** operator specifies the gray tint to use for filling paths. The *gray* operand must be a real number between 0.0 (black) and 1.0 (white).

gray **G**

The **G** operator is similar to the **g** operator, but specifies the gray tint to use for stroking paths. The *gray* operand must be a real number between 0.0 (black) and 1.0 (white).

cyan magenta yellow black **k**

The **k** operator is equivalent to the PostScript language **setcmykcolor** operator. It specifies the color to use for *filling* paths. Each operand must be a real number between 0.0 (minimum intensity) and 1.0 (maximum intensity). If the **setcmykcolor** operator is not defined by the PostScript interpreter (except in the case of creating separations), the Adobe Illustrator prolog defines it in terms of the original **setrgbcolor** operator by transforming the operands as follows.

```
red = 1 - min(1, cyan + black)
   green = 1 - min(1, magenta + black)
   blue = 1 - min(1, yellow + black)
```

The PostScript interpreter automatically performs the conversion from red, green, and blue to gray for a monochrome output device using the following formula.

```
gray = 0.3 × red + 0.59 × green + 0.11 × blue
```

cyan magenta yellow black **K**

The **K** operator is similar to the **k** operator, but specifies the color to use for stroking paths.

cyan magenta yellow black (name) gray **x**

The **x** operator defines a custom color for filling paths. The *cyan*, *magenta*, *yellow*, and *black* operands are interpreted in the same way as for the **k** and **K** operators. Adobe Illustrator treats the *gray* operand the same as for the **g**/**G** operators, and specifies the screen fraction of the custom color in the range 0.0 to 1.0. The *name* operand is a valid PostScript language string that names the custom color. For example:

```
0.45 0 0.25 0 (PANTONE 570 CV) 0 x
```

The first four operands are CMYK values. name is the name of the color; this is popped off in execution. The last operand (*gray*) is the tint value.

A user can specify the tint value in percentages from 0 to 100%. The value is scaled to the range of 0 to 1 and then subtracted from 1 to determine what is to be written out in the PostScript language call. A custom color's CMYK values are each multiplied by the tint value.

cyan magenta yellow black (name) gray X

The **X** operator is similar to the **x** operator, but specifies the custom color to use for stroking paths.

(patternname) p_x p_y s_x s_y *angle rf r k ka [a b c d* t_x t_y*]* **p**

The **p** operator specifies the pattern for subsequent fill operations. The *patternname* operand names the pattern as defined in the script setup sequence (see section section 5"). The remaining operands specify the transformations—in order—to be applied to the pattern before using it to fill a path.

p_x p_y Specify the offset from the ruler origin of the origin to be used for tiling the pattern. Each distance specified in points.

s_x s_y Specify the scale factors to be applied to the *x* and *y* dimensions, respectively, of the pattern.

angle Specifies the angle in counterclockwise degrees to rotate the pattern.

rf Flag indicating whether to apply a reflection to the pattern (1 = *true*, 0 = *false*).

r Specifies the angle of the line from the origin about which the pattern is reflected. Used if the *rf* operand is non-zero.

k Specifies the shear angle.

ka Specifies the shear axis.

[a b c d t_x t_y] Specifies the initial matrix to which all other pattern transformations are to be applied. This matrix describes transformations that are not otherwise expressible as the single combination of the other transformations. (See the description of the *Transform Pattern Style* sub-menu of the *Paint* menu in the *Adobe Illustrator 3.0 User Guide*.)

(patternname) p_x p_y s_x s_y angle rf r k ka [a b c d t_x t_y] **P**

The **P** operator is similar to the **p** operator, but specifies the pattern for use in stroking paths.

flag **O**

The **O** operator specifies whether to use overprinting when filling a path. If *flag* is 1 overprinting is used; otherwise *flag* must be 0. See *Adobe Illustrator 3.0 Color Guide* for a discussion of overprinting in Adobe Illustrator 3.x documents.

flag **R**

The **R** operator is similar to the **O** operator, but specifies whether to use overprinting when stroking a path.

6.4 Group Operators

Two operators support Adobe Illustrator’s ability to combine separate graphic elements into a single object.

– u

The **u** operator marks the beginning of a sequence of elements to be grouped into a composite object. All subsequent graphical elements in the script—including other groups, and up to a matching **U** operator—are included in the group.

– U

The **U** operator marks the end of a sequence of elements to be grouped into a composite object. A **u** operator must precede the **U** operator.

6.5 Paths

In the PostScript language, you draw by constructing a path and then filling or stroking it. This section defines the path operators. You can construct only one path (the *current path*) at a time. The current path is initially empty. The painting operators reset it to empty after execution.

6.6 Path Construction Operators

A path is constructed by appending segments which are either straight lines or Bézier curves. The last point on a segment is called the *current point*; new segments are always appended to the current point. The first operator on a path must be **m** to establish an initial current point.

As each new segment is appended to the path, Adobe Illustrator marks the new current point as either a smooth point or a corner point. If the point is smooth, Illustrator assumes collinearity of the point and the two associated Bézier direction points of the segments connected by the point. If the point is a corner point, there is no assumed constraint. You can think of a straight line segment as a "degenerate" Bézier curve in which the direction points are coincident with the end points.

The syntax for a path is as follows.

<path geometry) ::=	<m> {<path operator>}*
<path operator> ::=	<l>|<L>|<c>|<C>|<v>|<V>|<y>|<Y>
<path render> ::=	<N>|(closepath; no fill no stroke) <n>|(neither fill nor stroke) <F>|(fill) <f>|(closepath; fill) <S>|(stroke) <s>|(closepath; stroke) <B>|(fill and stroke) <b>(closepath; fill and stroke)

x y **m**

The **m** operator is equivalent to the PostScript language **moveto** operator. It changes the current point to *x, y*, omitting any connecting line segment. A path must have **m** as its first operator.

x y **l**

The **l** (lowercase L) operator appends a straight line segment from the current point to *x, y*. The new current point is a smooth point.

x y **L**

The **L** operator is similar to the **l** operator, but the new current point is a corner.

x1 y1 x2 y2 x3 y3 **c**

The **c** operator appends a Bézier curve to the path from the current point to x_3, y_3 using x_1, y_1 and x_2, y_2 as the Bézier direction points. The new current point is a smooth point.

x1 y1 x2 y2 x3 y3 **C**

The **C** operator is similar to the **c** operator, but the new current point is a corner.

x2 y2 x3 y3 **v**

The **v** operator adds a Bézier curve segment to the current path between the current point and the point x_3, y_3, using the current point and then x_2, y_2 as the Bézier direction points. The new current point is a smooth point.

x2 y2 x3 y3 **V**

The **V** operator is similar to the **v** operator, but the new current point is a corner.

x1 y1 x3 y3 **y**

The **y** operator appends a Bézier curve to the current path between the current point and the point x_3, y_3 using x_1, y_1 and x_3, y_3 as the Bézier direction points. The new current point is x_3, y_3 and is a smooth point.

x1 y1 x3 y3 **Y**

The **Y** operator is similar to the **y** operator, but the new current point is a corner.

6.7 Path Painting Operators

Each of the path painting operators consumes the *current path* and resets it to empty.

– **N**

The **N** operator neither fills nor strokes the current path, leaving it as an open path (see the **F/f** and **S/s** operators). Paths that are invisible in the final document may be used as templates, alignment marks, and so forth while using Adobe Illustrator to edit a document.

– **n**

The **n** operator is similar to the **N** operator, but first closes the current path.

– F

The **F** operator fills the area enclosed by the current path with the current filling color or pattern, leaving it as an open path. The inside of the current path is determined by the zero winding rule. See *PostScript Language Reference Manual, Second Edition* for “insideness” testing by the PostScript interpreter.

– f

The **f** operator is similar to the **F** operator, but first closes the current path.

– S

The **S** operator strokes the current path with a line using the current stroking color or pattern. The line width is specified by the graphics state (see the **w** operator) and the line is centered on the path with its sides parallel to the path. The joins between path segments are specified by the line join parameter in the graphics state (see the **j** operator), the ends of the path segments or dash lines within a segment (see the **d** operator) are specified by the end cap parameter of the graphics state (see the **J** operator).

– s

The **s** operator is similar to the **S** operator, but first closes the current path.

– B

The **B** operator is similar to the **F** operator, but both fills and strokes the path, leaving it as open.

– b

The **b** operator is also similar to the **f** operator, but both fills and strokes the path, leaving it as closed.

6.8 Compound Paths

Compound paths are equivalent to PostScript language paths; Illustrator paths are somewhat simpler. Letter shapes are often compound paths—for example, the letter A, because it has the enclosed counter.

The syntax for compound paths is:

```
<compound path> ::=  <*u>
                     <compound path element>+
                     <*U>

<compound path element> ::=
                     <path object>|<compound group>

<compound group> ::=
                     <u>
                     <compound path element>+
                     <U>
```

6.9 Clipping (Masking) Operators

Masks are conceptually similar to *ruby lith* and similar products in the graphic art industry—a red surface that blocks sections of a photo or other art work. An object used as a mask and the objects that it masks are bounded in the file by the **q** and **Q** operators as though they were grouped. Masks can be made from a path, a group of objects, or one or more compound objects. Each new mask intersects the current clipping path in the same way that the **clip** operator does.

```
<group with a mask> ::=
                     <q>
                     {<object>}*
                     {<masked object>}*
                     <Q>

<masked object> ::=  <mask>|<object>

<mask> ::=           <path mask>|<compound path mask>

<compound path mask> ::=
                     <*u>
                     <compound path mask element>+
                     <*U>

<compound path mask element> ::=
                     <path mask>|<compound mask group>

<compound mask group> ::=
                     <compound mask bottom group>|
                     <compound mask non-bottom group>
```

<compound mask bottom group> ::=
{<A>}
<q>
<path mask>+
<Q>

<compound mask non-bottom group> ::=
{<A>}
<u>
<compound mask group>+
<U>

– q

The **q** operator is similar to the **u** operator, except that an object in the group specifies a *mask* (clip path). The mask is a boundary for subsequent objects in the group, so that only objects within the boundary are visible when the illustration is rendered.

– **Q**

The **Q** operator is similar to the **U** operator, except that it marks the end of a sequence of elements containing a mask. A **q** operator must precede it.

– **H**

The **H** operator neither fills nor strokes the current path, but does not consume the path. This operator is used when establishing a mask.

– **h**

The **h** operator is similar to the **H** operator, but first closes the current path. This operator is used when establishing a mask.

– **W**

The **W** operator intersects the current clip path in the graphics state with the current path and sets a new, reduced clip path in the graphics state. No marks are made outside the area enclosed by the current clip path by subsequent fill and stroke operations until a **Q** operator appears. The **Q** operator restores the masking that was in effect at the matching **q** operator. This operator is used to establish a mask.

6.10 Text as Masks

You can use text objects as masks by using the **Tr** operator with a *rendermode* of 4 through 7. See section section 7.5, “Text Rendering Operators” for a full explanation of text rendering. When using text objects as masks, the entire text object becomes the mask. Illustrator cannot preview this kind of masking on the screen, although it prints properly. To see the effect of text used as a mask on the screen, convert the text to outlines.

7 Text

Illustrator writes out two kinds of information about text: *revisable* information and *final-form* printing information. Revisable information consists of those settings the user makes and can examine in Illustrator’s various dialog boxes—font size, for example. Final-form information is regenerated by Illustrator once the file has been read—it primarily helps to print text properly. Final-form information consists of such information as character positioning. This section differentiates between revisable, required information and final-form information.

Note *You can safely leave final-form information out of files and still have Illustrator read them correctly and regenerate the information; however, the file itself will not be an official EPS file and will not print.*

There are three kinds of text in Illustrator 3.x.

- *Point text* is created by clicking the text tool to create an insertion point, then typing. The first line of the text block begins at the insertion point. A carriage return determines the line break.

- *Area text* is created by clicking the area text tool and dragging a rectangle, and then starting to type. You can also create area text by clicking a path, selecting the area text tool, and then typing. A single text object can contain multiple text area elements.

- *Text on a path* follows the shape of a path. It is created by clicking an *open* path.

Text can *overflow* an area frame and thus not be visible on screen or when printed. Illustrator 3.x still saves such overflow text to the file, however. *Linked areas* allow text to flow from one area to another. Linked areas are part of the same text object.

Note *You cannot link point text to an area text object. All flowed text must appear in area texts.*

Area text objects can also be grouped with paths laying on top so that the text in the text objects “wraps” around (or avoids) the paths. The syntax for wraparound text is

<wraparound group> ::=
<*w>
{<object>}*
{<wraparound objects>}*
<*W>

<wraparound objects> ::=
<text object>|<object>

Wraparound text begins with the ***w** operator. Zero or more objects follow, then a standard Illustrator text object follows, and is in turn followed by zero or more Illustrator graphic objects around which the text will wrap. The wraparound text group ends with the ***W** operator.

Text can be *invisible* if you set its style to *neither* fill nor stroke. The PostScript interpreter does not render such text on the page. However, Illustrator writes the text to the file with the same structure as if the text were rendered.

7.1 Text Syntax

The syntax of text in Illustrator 3.x is as follows:

<text object> ::= <To>
<text at a point>|
<text area>|
<text along a path>
<TO>

<text at a point> ::= <Tp>
<TP>
<text run>+

<text area> ::= <text area element>+
{<overflow text>}

<text area element> ::=
<Tp>
<path object>
<TP>
<text run>+

<text along a path> ::=
<Tp>
<path object>
<TP>
<text run>+
{<overflow text>}*

<text run> ::= {<text style>|
<paint style>|
<text position>|
<Tk>}*
<text body>

<text style> ::= <Tr>|(render mode)
<Tf>|(font & size)
<Ts>|(rise and fall)
<Tz>|(horizontal scaling)
<Tt>|(tracking)
<TA>|(automatic kerning)
<TC>|(intercharacter spacing)
<TW>|(interword spacing)
<Ti>|(indents)
<Ta>|(alignment)
<Tq>|(hanging quotations)
<Tl>(leading)

<text position> ::= (printing only)
<Tc>|(computed intechar spacing)
<Tw>|(computed interword space)
<Tm>|(text matrix)
<Td>|(translate)
<T*>|(translate down)
<TR>(reset matrix; found only in pattern prototypes)

<text body> ::= <Tx>|<Tj>|<T+>|<T–>

<overflow text> ::= {<text style>|<paint style>|<TK>}*
<TX>|<T+>

7.2 Text Attributes

A *text attribute* is a quality of the text such as font, justification, stroke, or fill. There are three kinds of text attribute used in Adobe Illustrator 3.x: character style, paragraph style, and paint style.

A *character attribute* is an attribute such as font size or scale that pertains to one or more characters. A*paragraph attribute* pertains to one or more paragraphs, and includes details such as justification and indentation. A *paint style attribute* pertains to any set of characters that all have the same paint style and character attributes (called a *text run*). You can apply any kind of paint style to characters—stroke or fill, stroke width changes, dashed lines, and so forth. In Illustrator, you set character and paragraph attributes in the Type Style dialog box.

7.3 Text Object Operators

A text object is bracketed by the **To** and the **TO** operators. The **To** operator begins a text object, and the **TO** operator ends it. Between the two can appear a series of operators that control the text path(s), set the text matrix, and set various other text attributes. Attributes are *modal*, that is, once an attribute operator has made a change, that change stays in force until changed again. While there is no special order in which the operators must appear, because of modality, operator order may have a bearing on the end result. See section section 7.1, "Text Syntax" for information about operator order.

A *text path* is a combination of the current transformation matrix and the path geometry of a *text container*—an area or object such as a box, circle, or other shape.

Note *If the Illustrator file includes more than one path between the* **To** *and* **TO** *delimiters, it must be area text; it means that text can flow from one object to another.*

7.4 Text Path Operators

Within a text object, each text path is bracketed by the **Tp** and **TP** operators. The **Tp** operator takes as its arguments a transformation matrix and a start point. The **Tp** and **TP** operators appear in every kind of text object. Its purpose is to provide a matrix for the text object and its container (if any). The operators vary in their use depending on text type.

- *Point text.* There is no path geometry. The **Tp** operator simply passes the matrix that is associated with the object.

<text at a point> ::= <Tp>
<TP>
<text run>+

```
1 0 0 1 116 527 0 Tp
TP
```

- *Text on a path*. There is no path geometry for the text. The entire text path is rotated (skewed, etc.). The actual path is bracketed inside **Tp** and **TP**. The *startPt* parameter is used only for text on a path. It indicates where the text starts on the path by giving the fractional number of the Bézier segment it starts on begining with 0. For example, text that starts in the center of the second Bézier segment would have a *startPt* of 1.5.

<text along a path> ::=
<Tp>
<path object>
<TP>
<text run>+
{<overflow text>}*

```
1 0 0 1 114 479 0 Tp
243 403 m
243 479 L
114 479 L
114 403 L
243 403 L
n
TP
0 -10.875 Td
0 Tr
0 O
0 g
(The quick brown fox ) Tx
0 -14.5 Td
(jumps over the lazy ) Tx
0 -14.5 Td
(dog.  The quick brown ) Tx
0 -14.5 Td
(fox jumps over the lazy ) Tx
0 -14.5 Td
(dog.  ) Tx
(\r) TX
```


- *Area text*. Each container that text flows into has its own **Tp** and **TP** pair and matrix associated with it. The container also includes information about stroke and fill for the container.

<text area> ::= <text area element>+
{<overflow text>}

<text area element> ::=
<Tp>
<path object>
<TP>
<text run>+

```
1 0 0 1 -342 1044 1.5104 Tp
109.5 309.5 m
117.8551 317.4372 116.4004 328.658 119.8602 338.3717 c
122.5416 345.9002 128.599 351.0478 134.564 356.4291 c
161.757 380.9608 206.5173 380.4135 231.1035 353.1425 c
246.4752 336.0923 253.1938 311.7104 275.641 302.8574 c
318.7067 285.8727 367.8791 298.6578 406.5 321.5 C
406.1633 321.5 405.835 321.5 405.5 321.5 c
```

```
N
TP
0.7152 0.6989 -0.6989 0.7152 126.0945 348.4862 Tm
0 Tr
0 O
0 g
(T) Tx
0.7435 0.6687 -0.6687 0.7435 131.317 353.508 Tm
(h) Tx
0.795 0.6066 -0.6066 0.795 136.1212 357.8995 Tm
(e) Tx
0.8733 0.4871 -0.4871 0.8733 143.9726 363.6125 Tm
(q) Tx
0.9148 0.4038 -0.4038 0.9148 149.6282 366.7758 Tm
(u) Tx
0.9388 0.3444 -0.3444 0.9388 155.631 369.3596 Tm
(i) Tx
0.9575 0.2886 -0.2886 0.9575 158.0023 370.2853 Tm
(c) Tx
0.9776 0.2105 -0.2105 0.9776 163.5868 371.975 Tm
(k) Tx
0.9965 0.0829 -0.0829 0.9965 172.4898 373.7108 Tm
(b) Tx
0.9999 0.0118 -0.0118 0.9999 178.9926 374.2097 Tm
(r) Tx
0.9982 -0.0598 0.0598 0.9982 182.8403 374.3088 Tm
(o) Tx
0.9865 -0.1638 0.1638 0.9865 189.295 373.9865 Tm
(w) Tx
0.9633 -0.2685 0.2685 0.9633 197.6105 372.5656 Tm
(n) Tx
0.924 -0.3824 0.3824 0.924 206.9198 369.6766 Tm
(f) Tx
0.893 -0.4501 0.4501 0.893 209.8895 368.5214 Tm
(o) Tx
0.8455 -0.5339 0.5339 0.8455 215.6646 365.6044 Tm
(x) Tx
0.776 -0.6307 0.6307 0.776 223.1863 360.5216 Tm
(j) Tx
0.7277 -0.6859 0.6859 0.7277 225.1888 358.9963 Tm
(u) Tx
0.6375 -0.7705 0.7705 0.6375 229.9913 354.5899 Tm
(m) Tx
0.5733 -0.8194 0.8194 0.5733 236.1842 346.9666 Tm
(p) Tx
0.5446 -0.8387 0.8387 0.5446 239.996 341.486 Tm
(s) Tx
0.5393 -0.8421 0.8421 0.5393 245.0267 333.6389 Tm
(o) Tx
0.5628 -0.8266 0.8266 0.5628 248.7057 327.8809 Tm
(v) Tx
0.6097 -0.7926 0.7926 0.6097 252.2511 322.6429 Tm
(e) Tx
0.6677 -0.7444 0.7444 0.6677 256.6117 317.0635 Tm
(r) Tx
0.7644 -0.6448 0.6448 0.7644 262.2055 311.2463 Tm
(t) Tx
0.838 -0.5456 0.5456 0.838 265.22 308.6137 Tm
```

```
(h) Tx
0.9226 -0.3857 0.3857 0.9226 271.7556 304.4719 Tm
(e) Tx
0.9542 -0.2992 0.2992 0.9542 281.684 300.6715 Tm
(l) Tx
0.9667 -0.2561 0.2561 0.9667 284.4584 299.7607 Tm
(a) Tx
0.9802 -0.198 0.198 0.9802 291.2396 297.9861 Tm
(z) Tx
0.9896 -0.144 0.144 0.9896 297.4289 296.7456 Tm
(y) Tx
0.9982 -0.0602 0.0602 0.9982 307.1617 295.4818 Tm
(d) Tx
1 -0.0054 0.0054 1 314.1365 295.0711 Tm
(o) Tx
0.9989 0.0468 -0.0468 0.9989 321.1079 295.0431 Tm
(g) Tx
0.9964 0.0842 -0.0842 0.9964 327.9832 295.4014 Tm
(.) Tx
(The quick brown fox jumps over the lazy dog.\r) TX
```

A text area element within a text object may be arbitrarily scaled, skewed, rotated, and translated using matrix operators. Text attributes such as whether the text is filled or stroked, its kerning and leading are controlled by attribute operators.

Adobe Illustrator has three text matrix handling operators that control where to place the text object. For text on a path, the **Tm** operator sets the text matrix. The **Td** and **T*** operators translate the text matrix to the start of the next line of text.

However, the three text matrix operators are final-form. Illustrator writes this information when it saves a file but ignores it when reading the file.

7.5 Text Rendering Operators

Text rendering mode is set by a call to the **Tr** operator. This operator takes as its argument one of the selectors in Table 1. The **Tx** and **Tj** operators actually render the text.

Table 1 *Text rendering modes*

Selector	*Rendering Mode*
0	fill text
1	stroke text
2	fill and stroke text
3	text with no fill and no stroke ("invisible")
4	mask and fill text
5	mask and stroke text
6	mask, fill, and stroke
7	mask (only) text
8	filled text followed by rendertype 9 (pattern prototype only)
9	stroked text preceded by render mode 8 text (pattern prototype only)

7.6 Kerning

Illustrator 3.x offers two types of kerning. These are:

- *Track kerning* (**Tt**). The specified amount of space is added between each pair of characters. Space is measured in thousandths of an em.

- *Pairwise kerning* (**TA**, **Tk**, **TK**). The **TA** (automatic kerning) operator tells Illustrator to turn automatic kerning on and use the kerning pairs specified in the font program itself. The **Tk** operator causes a manual or automatic kern—Illustrator puts a **Tk** in the file where kerning is necessary. Manual kerning overrides automatic kerning. Measurements that come from the kerning dialog box are expressed in thousandths of an em. The **TK** operator is identical to the **Tk** operator, but applies to overflow text that is not displayed or printed.

7.7 Spacing

Spacing control operators apply to justified and non-justified text. In the case of non-justified text, only the optimum value is used.

- *Word spacing* (**TW**). Specifies the minimum, optimum, and maximum space between words (space characters). Measurement is in a percentage of the width of a regular space character.

- *Character spacing* (**TC**). Specifies the minimum, optimum, and maximum space between characters in the file. Measurement is in a percentage of the width of a regular space character.

Two other spacing operators also appear in Illustrator files. They are final-form.

- *Computed word spacing* (**Tw**). This is the actual word spacing for a particular line of text. It is recomputed on a per-line basis.

- *Computed character spacing* (**Tc**). This is the actual character spacing for a particular line of text. It is recomputed on a per-line basis.

7.8 Line Spacing and Discretionary Hyphens

Line and paragraph leading is set using the **Tl** (lowercase L) operator, in units of one thousandth of an em-square. Superscripting and subscripting are set using the **Ts** operator with (respectively) positive and negative values expressed in points.

The **T+** operator appears whenever the user has inserted a discretionary hyphen.

The **T-** operator appears in text wherever a discretionary hyphen prints.

7.9 Alignment and Justification

A group of characters that share the same size, paint style, tracking, and so forth can be written out as a single call to **Tx** (for non-justified text) or **Tj** (for justified text).

The **Ta** operator sets text alignment: left, centered, right, or justified.

The **Ti** (lowercase i) operator controls line indentation. It takes arguments that state the values of the left indent, a delta from the left indent for the first line, and a right indent in points.

The **Tz** operator condenses or expands the horizontal scaling of a character as a percentage of the regular font size.

7.10 Text Operators

The following list of text operators includes whether the operator is revisable or final-form. **R** signifies a revisable operator, **F** signifies a final-form operator. Illustrator recalculates the values of all such final-form operators after reading the file.

type **To**

[Revisable]. This operator begins a text object. The type argument can be one of:

0point text
1area text
2path text

TO

[Revisable]. The **TO** operator ends a text object and restores the current transformation matrix.

[a b c d tx ty] startPt **Tp**

[Revisable]. This operator brackets the text path. It concatenates the matrix parameter with the current transformation matrix (CTM). For text on a path only, the *startPt* operand indicates where the text starts on the arc length of the path by giving a fractional number that signifies its position along the Bézier segment on which the path starts, begining with 0. For example, text that starts in the center of the second Bézier segment would have a *startPt* of 1.5.

TP

[Revisable]. The **TP** operator ends the text path.

Matrix Operators

[a b c d tx ty] **Tm**

[Final-Form]. The **Tm** operator sets the text matrix for text along a path.

tx ty **Td**

[Final-Form]. This operator translates the text matrix by t_x and t_y. to the beginning of the next line of text.

T*

[Final-Form]. The **T*** operator translates the text matrix by *–lineleading, 0* to the beginning of the next line of text.

[a b c d tx ty] **TR**

[Final-Form]. The **TR** operator resets the pattern matrix for the pattern prototype only. See section section 5.4, “Pattern Definition” for more information about patterns.

Text Attribute Operators

render **Tr**

[Revisable]. The **Tr** operator sets the render mode for any text that follows. The **Tx** and **Tj** operators actually render the text. The *render* argument can be one of the following values:—

0—fill text
1—stroke text
2—fill and stroke text
3—invisible text
4—mask and fill text
5—mask and stroke text
6—mask, fill, and stroke text
7—mask (only) text
8—filled text followed by render mode 9 (pattern prototype only)
9—stroked text (preceded by render mode 8 text, pattern prototype only)

Modes 4 through 7 are used within pattern definitions when text, as part of the pattern, is filled *and* stroked with two different colors. Modes 8 and 9 are used in pattern prototypes that contain text objects.

fontname size **Tf**

[Revisable]. The **Tf** operator specifies the reencoded name of the font to use and its size in points.

alignment **Ta**

[Revisable]. The **Ta** operator sets text alignment both horizontally and vertically. The value for *alignment* can be one of the following:

0—left aligned
1—center aligned
2—right aligned
3—justified (right and left)
4—justified including last line

leading paragraphLeading **Tl**

[Revisable]. The **Tl** (lowercase l) operator sets the leading for the paragraph. The *leading* argument sets the leading between lines within a paragraph and the *paragraphLeading* argument sets the extra leading between paragraphs.

userTracking **Tt**

[Revisable]. The **Tt** operator sets additional space to add between characters in units of one thousandth of an em.

minSpace optSpace maxSpace **TW**

[Revisable]. The **TW** operator sets word spacing, where the minimum, optimum, and maximum space between words (the size of space characters) is expressed as a percentage of the width of a regular space character: 100% equals the width of one space character.

wordSpace **Tw**

[Final-Form]. The **Tw** operator sets computed word spacing.

minSpace optSpace maxSpace **TC**

[Revisable]. The **TC** operator sets character spacing, where the minimum, optimum, and maximum space between characters is expressed as a percentage of the width of a regular space character: 100% equals the width of one space character.

charSpace **Tc**

[Final-Form]. The **Tc** operator sets computed char spacing.

rise **Ts**

[Revisable]. The **Ts** operator sets the distance by which a character is raised above the baseline in superscripting and dropped below the baseline when subscripting. The argument *rise* is expressed in points.

firstStartIndent otherStartIndent stopIndent **Ti**

[Revisable]. The **Ti** operator sets the indentation of a paragraph. The argument *firstStartIndent* specifies the indentation for the left side of the paragraph, *otherStartIndent* specifies a delta from the left side that applies only to the first line, and *stopIndent* specifies any right side indentation. The three arguments are specified in points.

scalePercent **Tz**

[Revisable]. The **Tz** operator sets horizontal scaling of a line of text by condensing or expanding the line in terms of a percentage of the normal font width.

autoKern **TA**

[Revisable]. The **TA** operator specifies whether to use pairwise kerning. Illustrator uses the kerning pairs built into the Type 1 font program. If the *autoKern* argument is 0, Illustrator uses no pair kerning; if the *autoKern* argument is 1, Illustrator does use pair kerning.

hangingQuotes **Tq**

[Revisable]. The **Tq** operator determines whether a run of text uses hanging quotation marks; a hanging quotation mark is one that extends beyond the left or right edge of the text block. If the *hangingQuotes* argument is 0, Illustrator does not use hanging quotation marks; if the argument is 1, it does. For the domestic version of Adobe Illustrator, the marks which hang are: period, comma, semicolon, colon, backquote, left and right single quotes, left and right double quotes, dash, en-dash, em-dash. That is:

. , ; : ` ‘ ’ “ ” - – —

Note *Marks may be different for international editions.*

Text Body Operators

textString **Tx**

[Revisable]. The **Tx** operator holds a string to be used as the text of a path object. The string is not justified. The *textString* argument is any series of ASCII codes in the native platform encoding; the character combination \r signifies a new paragraph.

textString **Tj**

[Revisable]. The **Tj** operator renders a string of justified text. The *textString* argument is any series of ASCII codes.

textString **TX**

[Revisable]. Text that overflows the text area (invisible). Equivalent to the **Tx** operator.

autoKern kernValue **Tk**

[Revisable]. The **Tk** operator indicates that kerning takes place. A value of 0 or 1 for *autoKern* tells Illustrator whether to use manual (0) or automatic (1) kerning. The *kernValue* operand sets a kerning value in units of one thousandth of an em. When *autoKern* is 0, Illustrator uses the value in *kernValue*; when *autoKern* is 1, Illustrator uses the built-in kerning pairs.

autoKern kernValue **TK**

[Revisable] The **TK** operator is the same as the **Tk** operator, except that it applies to overflow text (invisible).

T+

[Revisable]. The **T+** operator marks every instance of a discretionary hyphen.

T-

[Revisable]. The **T-** operator marks where a discretionary hyphen has been printed.

8 Guide Operators

Guides act as control lines or shapes, and are similar to the “grid” feature in some drawing programs. They do not print and do not show up during preview, but other objects “snap to” guides for positioning. Guides can be lines or objects.

When you turn an object into a guide, it retains its color and other attribute information. If and when you release the object from being a guide, its attributes are restored.

Adobe Illustrator 3.x writes guides out to the file in the form of a path. The * operator begins and ends a path used as a guide. The * operator takes a string parameter which is one of the path render operators, for example **(F)***. The default form for guides created via the ruler is **(N)***. For example:

```
(N) *
  315 1044 m
  315 -252 L
  (N) *
```

Guides can appear within groups, and guides themselves can be groups.

9 Placed Art Operators

The syntax for placing an EPS file into an Adobe Illustrator document is

<placed art object> ::=	
	<’> <art reference> <~>
<art reference> ::=	<file reference>|<file inline>
<file reference> ::=	%%IncludeFile: <filename>
<file inline> ::=	%%BeginDocument:<filename> … included file contents %%EndDocument
<filename> ::=	platform-specific path name of file
<subscriber object> ::=	
	%AI3_Subscriber:<subscriber ID> <placed art object>
<subscriber ID> ::=	resource number of SECT resource in file

The **%%subscriber** comment concerns the Macintosh publish and subscribe facility available in System 7. It reads in an edition which has been published as graphics. The *section ID* shows the resource ID of the section (in a *sect* resource), as stored in the file. See *Inside Macintosh Volume VI* for more information on publish and subscribe. If you need to use this feature, you will also need to store the bounds in a *bnds* resource of the given id (whose format is a rectangle of Fixed values: left, top, right, bottom), and the section options in a *psop* resource of the given id (whose format is a two-byte integer, value 0).

The filename in the **%%IncludeFile** comment must be the same as that specified for the **‘** (single tick-mark) operator. This includes a file *by reference* in a given Illustrator document.

You can also save included files *directly* (rather than by reference) in an Adobe Illustrator document. If you wish to do that, replace the **%%IncludeFile** comment with the structure:

```
%%BeginDocument: filename
  … put included file here
  %%EndDocument
```

Adobe Illustrator automatically modifies the **%%DocumentFonts** comment of the including document to add any fonts in the **%%DocumentFonts** comment of an included document.

[a b c d t_x t_y] llx lly urx ury (filename) **‘**

The **‘** (single quote or tick-mark) operator specifies that the document stored in *filename* is to be imported into the illustration. The imported file is assumed to be an EPS-conforming document.

The *filename* string is the full pathname for the file in the operating system’s file system. Adobe Illustrator concatenates the matrix operand with the current transformation matrix to establish a new user space and an origin for the imported document. See *PostScript Language Reference Manual, Second Edition* for more information about transformation matrices. The matrix handles any rotation and reflection to be applied to the imported document. The *llx lly urx ury* operands specify the bounding box from the imported document as stated by the **%%BoundingBox** comment in the imported document’s prolog. In addition to establishing a new user space, the **‘** operator does the following (or their equivalents):

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

– ~

The **~** operator restores the user space in effect when the preceding **‘** operator was executed.

10 Graphs

Illustrator’s *graphs* capability builds business graphs, such as bar (column) graphs and pie charts. You can build such graphs using Illustrator’s general drawing abilities, but having Illustrator itself build the graphs from data you enter or import helps assure accuracy.

The **Gs** operator begins a graph and the **GS** operator ends it. All graph operators consist of two characters and begin with **G**.

A graph in an Illustrator file has two parts: the *functional specification* and the *objects* that make up the graph. The functional spec acts like an internal header; it contains information about the graph, such as graph type, axis parameters, and graph data values. The graph objects are regular Illustrator path and text objects, which allows the file to be incorporated as an EPS file in other files for documents.

On command, Illustrator can recreate a graph based only on the functional spec. However, on reading an Illustrator file that contains a graph, Illustrator *does not* automatically recalculate the graph. Instead, it displays the graph objects as they appear in the file—exactly as Illustrator would treat any other objects in a file.

Note *When preparing files that contain graphs for use with Illustrator, you must make sure that the functional spec matches the graph objects (if you choose to supply them). Otherwise, recalculating will result in changes to the graph.*

10.1 Parts of a Graph

holds labeled illustration
Those parts are:

axis
label group
axis tick group
category axis group
edge
legend group
data column
series 0
series 1

10.2 Syntax

The syntax of the graph section is:

```
<graph object> ::=      <Gs>
                        <graph functional spec>
                        {<graph customizations>}
                        <graph group object>
                        <GS>

<graph functional spec> ::=
                        <graph size and dialog values>
                        {<graph subscriptions>
                        <graph axis>
                        <graph axis>
                        <graph axis>
                        <graph table specs>

<graph size and dialog values> ::=
                        <Gb>
                        <Gy>
                        <Gd>
```

<graph axis> ::= <Ga>
<GA>

<graph table specs> ::=
{<Gw>}*
<Gz>
<Gc>+
<GC>

<graph customizations> ::=
<Gt>
{<graph customization>}*
<GT>

<graph customization> ::=
{<graph customization operator>}*
<GX>|<Gg>
{<Gv>}

<graph customization operator> ::=
{<Gm>}
{<Gf>}
{<Gy>}
{<GD>}
{<Ge>}
{<G1>}(gee-one)
{<Gi>}
{<Gl>}(gee-ell)
{<Gp>}
{<Gx>}
{Gr>}
{<G+>}
{<Gg>}
{<A>}
{<paint style>}*
{<text style>}

<graph subscriptions> ::=
<Gj>

<graph group object> ::=
<u>
{<graph rendered object>}*
<U>

<graph rendered object> ::=
<object>|<graph group object>
{<Go>}

10.3 Functional Specification

The functional specification acts as an internal header. It contains information about the bounds of the graph, its style, its axes, and the data that make it up. From the functional spec, Illustrator can reconstruct the graph (Illustrator does not automatically recalculate the graph on reading in a file); in fact, Illustrator can reconstruct a graph from no more than the functional spec and does not require graph objects at all.

Statements in the functional spec take the form of comments, and begin with the characters **%_**. This allows an Illustrator file with a graph in it to be used as an EPS file, in which comments are not executed. The following example shows a simple functional spec.

```
%_Gs
%_548 122 301 440 Gb
%_5 0 0 14 1 1 0 0 3 44 Gy
%_7 2 1 0 90 80 0 Gd
%_1 () Ga 0 14 0 1 0.2 0 1 () GA
%_2 () Ga 0 14 0 5 1 0 0 () GA
%_4 () Ga 0 14 0 5 1 0 0 () GA
%_4 4 1 1 Gz
%_() Gc (FIRSTCOL) Gc (SECONDCOL) Gc (THIRDCOL) Gc (GLOVES) Gc 1 3
   1 (BALLS) Gc 2 2 5 (BATS) Gc 3 4 2 Gc GC
```

The **%_Gs** operator opens the graph section. The next operator, **Gb**, defines the bounds of the graph and determines where Illustrator draws the graph's axes. **Gy** and **Gd** apply some of the values from Illustrator's Graph Style dialog box. The three **GA** operators control how values appear on the axes. The **Gz** operator works with the **GA** operator to specify the graph axis.

The **Gc** operator reads in cell values, one by one, and puts them in the cell data table—rows and columns like a spreadsheet—from which Illustrator constructs its graphs. Operators in Illustrator are limited to one string parameter and a maximum of 16 parameters overall; consequently, there is usually more than one **Gc** operator to a line. The cells in the series read from left to right in a row; then the next row down is read in. String values appear within parentheses. If a cell holds no value, the file holds its place with a pair of empty parentheses. In the example

```
%_() Gc (FIRSTCOL) Gc (SECONDCOL) Gc (THIRDCOL) Gc (GLOVES) Gc 1 3
   1 (BALLS) Gc 2 2 5 (BATS) Gc 3 4 2 Gc GC
```

the contents of row1 column1 is an empty string. Because it must be allowed to hold a string, the **Gc** operator follows it (one string allowed per operator). The contents of row1 column2 is the string *FIRSTCOL*. Again, because it is a string, the **Gc** operator must immediately follow it. The **Gc** parameter doesn't "care" about the coordinates of the parameters it sets, only their number. For example, if there were 40 numerical data points to place in the table, the first **Gc** can place 16, the second **Gc** can place 16, and the third **Gc** can place eight—with no regard to where those 40 data points fall on the table. It is the *order* in which they occur within the file that determines their row and column coordinates.

If you are preparing a file to be used with Illustrator, label and data point order is critical to creating an accurate graph.

The final **GC** ends the data table.

Note *Illustrator notes any edits you make to the graph after you build it initially; it specifies those edits as a series of changes following the functional spec. These customizations are covered in section section 10.5.”*

10.4 Operators in the Functional Spec

Gs

This operator signals the beginning the graph object. Must be the first operator in this graph object. It takes no parameters.

GS

This operator signals the end the graph object. It takes no parameters.

left top right bottom **Gb**

The **Gb** operator shows the *bounds* of the graph. The parameters are decimal numbers, in the same coordinate system as other objects. This rectangular area defines where Illustrator draws the axes' lines. The axis labels, category labels, legend boxes, and legend labels are all slightly outside this rectangle. The axis lines are drawn exactly on the edges of this rectangle. This is a pre-customization rectangle. See section section .”

graphType shadow dataPaintOrder pieLegendStyle drawMarks drawLines drawLinesAsShapes drawLegendsAcrossTop lineShapeWidth whichAxis [piePercentage piePctDigits] **Gy**

This operator defines values for the Graph Style dialog box, just as does **Gd**; however, these values can be applied to individual series in a graph, overriding, for that series, the default graph style.

graphType — This is the style of graph, for the whole graph (which may be overridden in any number of series), or for one series. The values for *graphType* are:

5—grouped column graph
6—stacked column graph
7—line graph
8—pie chart
9—scatter graph
10—area graph

Note — *The values for scatter and area are in the opposite order from the dialog box and tools.*

shadow — This controls whether to create an object showing a shadow when creating the graphical objects. 1 means draw the shadow. Values: 0 or 1. Default: 0.

dataPointOrder — When individual items in a data series overlap graphically, the items corresponding to the upper rows in the cell table are covered by the graphical objects corresponding to the lower rows in the cell table. A value of 1 does the opposite. See the *seriesPaintOrder* parameter in the **Gd** operator for more information.

pieLegendStyle — Legends in pie charts can be numerous types; they can be the same as for bar and line graphs (boxes along the right or top edge, with labels next to them), a label within each wedge, or none at all. The values are:

14 same as bar/line graphs
15 legends in wedges
16 no legends

drawMarks — When creating line and scatter graphs, Illustrator can place small marks at the data points or have lines go through them without special marks. This parameter controls the existence of the marks. 1 means draw marks. Values: 1 or 0. Default: 0.

drawLines — When creating line and scatter graphs, Illustrator can draw lines that connect the data points, or not. This parameter controls the existence of the lines. 1 means draw them. Note that if this and the above value are both zero, Illustrator draws nothing within the data area. Values: 1 or 0. Default: 1.

drawLinesAsShapes When creating line and scatter graphs, Illustrator can use simple lines to represent the data, or use shapes that represent thick lines. 1 means draw the shapes. Values: 1 or 0. Default: 0.

Note that if *drawLines* is 0, this parameter is ignored (but still must be supplied).

drawLegendsAcrossTop When creating a graph that contains legends, Illustrator can either put the legends on the right edge of the graph, going down, or on top, going left-to-right. 1 means put them across the top. Values: 0 or 1. Note that this is a parameter that applies to the graph as a whole and cannot be applied to individual series (although a value, as a placeholder, must appear here). Default: 0.

lineShapeWidth If *drawLinesAsShapes* is on, this is the width (in points) of the shape that is created. Values: 0.0 to 100.0. Default: 6.

whichAxis This shows which axis (left or right) against which to measure the data. Often, graphs are created that show trends of two series that have very different ranges (for instance, net income in millions of dollars against share price under one hundred dollars). One axis can show values in the millions while the other shows the values under 100. The values are:

44 Use left axis
45 Use right axis

piePercentage This is a flag that tells Illustrator to display a percentage number inside a pie wedge. It is available only for Adobe Illustrator for Windows Version 4.x. A value of 1 tells Illustrator to display the percentage; 0 tells it not to display the percentage.

piePctDigits This value tells Illustrator the number of digits to place after the decimal point when displaying a percentage in a pie wedge. It is available only for Adobe Illustrator for Windows Version 4.x.

colWidth cellTableDecimalPrecision seriesPaintOrder useBothAxis barWidthPercentage groupWidthPercentage drawLinesEdgeToEdge **Gd**

This operator defines some of the values in the Graph Style dialog box. Each parameter is defined following. The values for this operator apply to the entire graph; they cannot be applied to individual series in a graph (a series corresponds to a column in the cell table).

colWidth — This is the default column width of the cell table for this graph, expressed as the number of characters that will fit in that column. Inidividual column widths can be overwritten by the **Gw** operator. Range: 3 to 20.

cellTableDecimalPrecision — This is the number of decimal places shown in the cell table. The values are kept at full precision; only the display of those values is affected. Range: 0 to 10.

seriesPaintOrder — When Illustrator creates a graph, it creates the objects for each series. By default, it puts the each subsequent series in front of the previous one: that is, if they overlap, graphical objects representing later series will display and print on top of earlier series. A value of 1 displays and print the series in the opposite order; that is, the first series will be frontmost in the graph and will paint over other series if they overlap. Note that this does not change the position of the series; only the painting order. Values: 0 or 1; default value 0.

useBothAxis — The user can choose either left, right, or both axes to display, and independently set the axis numbers and tick placement for each axis. A parameter value of 1 “copies” the values from one axis to the other. Which axis is copied depends on which axis has been chosen as the default for all series (both may have been chosen, in which case Illustrator ignores this parameter). If one axis has been chosen, the other axis is copied. Values: 0 or 1. Default: 0.

barWidthPercentage — This is the percentage of the width available for a bar that will be used for that bar. (In the Illustrator manual, vertical-bar graphs are called “column graphs.” Use of the term “bar” distinguishes the graphic rectangle that represents one piece of data in the cell table from a column of cells in the table). Illustrator divides the available width of a graph into portions depending on how many series there are and how many individual bars there are per series. For example, if this parameter value is 100, for grouped-column graphs there would be *no* space at all between individual bars in a group. The lower the percentage, the more space between individual bars. The higher, the less—values over 100, which are legal, causes bars to overlap. See the *dataPaintOrder* parameter of **Gy** for how to influence the paint order of individual overlapping bars. For stacked-column graphs, individual bars are all in the same stack—the *barWidthPercentage* and *groupWidthPercentage* are multiplied to obtain the real percentage to use for each stack. If both are 100, no space shows between stacks. One value of 64 and the other of 100, versus both values of 80, shows the same graph for stacked-column graphs but

DRAFT

shows different group *and* bar spacing for grouped-column graphs. Range: 1.0 to 1000.0 (that is, one-hundredth of the available width to ten times the available width).

groupWidthPercentage — This is the percentage of the width available for a group (in grouped-column graphs) or a single stack (in stacked-column graphs) that will be used for that group or stack. See *barWidthPercentage*, above, for a fuller description. Range: 1.0 to 1000.0 (that is, one-hundredth of the available width to ten times the available width).

drawLinesEdgeToEdge — In line graphs, the user can decide whether to leave a little space between the left and right axes and the beginning and end of the data lines. Usually this is done to leave room for labels at the bottom of the graphs. This parameter defines whether or not to draw the lines right to the edges. Note that this widens the area available for each label along the bottom, because the same number of labels now have a slightly wider area to draw. 1 means draw all the way to the edge. Values: 0 or 1. Default: 0.

leftColumn topRow rightColumn rightRow sectionID **Gj**

This operator concerns the Macintosh publish and subscribe facility available in System 7. It reads in an edition which has been published as text. The bounds (0 is the first column) show how big the section is, although that is ignored: if the read-in section is larger or smaller than the bounds, they're reset by Illustrator. The *sectionID* shows the resource ID of the section (in a *sect* resource), as stored in the file. See *Inside Macintosh Volume VI* for more information on publish and subscribe. If you need to use this, you will need to store the *sectionHandle* in the *sect* resource with the given id, the bounds in a *bnds* resource of the given id (whose format is a rectangle of Fixed values: left, top, right, bottom), and the section options in a *psop* resource of the given id (whose format is a two-byte integer, value 0).

whichAxis beforeString **Ga**

This is the beginning of the specification of a graph axis. It continues and ends with the **GA** operator.

whichAxis — This tells which axis to set to this specification. The values are:

1 Bottom axis
2 Left axis
4 Right axis

beforeString — This string is appended to the label next to each axis tick mark; a value of *(units)* shows as "100 units" rather than as just 100. These labels show the numerical representation of the tick, which can include a currency symbol or some other representation *before* the value. If the value of *beforeString* is $, the axis label becomes $100 instead of 100. Length: up to 9 characters. Note that any characters other than standard ASCII must be entered as an octal number with a slash (this is a PostScript standard), so £ and ¥ must be decoded into their ASCII values.

DRAFT

useManualValues tickMarkType minimumValue maximumValue betweenValue smallTicksPerValue drawMarksBetweenLabels afterString **GA**

useManualValues — The user can choose to have Illustrator decide which axis values are appropriate given the range of the cell data, or the user can specify his own values (for the current axis, specified by the **Ga** operator). 1 means use manual values. Values: 0 or 1. Default: 0.

tickMarkType — The user can specify short tick marks, tick marks that stretch to the other edge of the graph, or none. The values are:

13—no tick marks
14—short tick marks
15—long tick marks

minimumValue — If the user has specified manual values for the axis, this parameter supplies the lower of the two values. Depending on the *betweenValue*, *minimumValue* may correspond to the topmost or bottommost tick mark. Range: any decimal number.

maximumValue — If the user has specified manual values for the axis, this parameter supplies the higher of the two values. Depending on the *betweenValue*, *maximumValue* may correspond to the topmost or bottommost tick mark. Range: any decimal number.

betweenValue — If the user has specified manual values for the axis, this parameter supplies the numerical difference between subsequent tick marks. If this number is positive, then *minimumValue* shows up on the bottom of the *y*-axis (on the left in the case of the *x*-axis). If this number is negative, then *maximumValue* shows up on the bottom of the *y*-axis (on the left in the case of the *x*-axis). Range: any decimal number.

smallTicksPerValue — The user may want a graph with tick marks every 10, but labels every 20. Illustrator uses *smallTicksPerValue* to divide *betweenValue* into smaller segments and display extra tick marks without displaying labels. Illustrator divides the betweenValue by this number, and zero is a valid result meaning none. This means that 0 and 1 do the same thing: produce no small ticks. Range: integers between 0 and 1000. Default: 0.

drawMarksBetweenLabels — This applies only to the *category axis*. Sometimes the user wants tick marks on the category axis to line up exactly with the labels (usually in the case of line and area graphs), and sometimes the user wants ticks between the labels (usually in the case of stacked-column and grouped-column graphs). If 1, the category axis draws tick marks between the labels. Values: 0 or 1. Default: 0.

afterString — As with *beforeString* in the **Ga** operator, this string is appended to the label next to each axis tick mark; a value of *(units)* shows as “100 units” rather than as just 100. Length: up to 9 characters. See *beforeString* for character value limitations.

rows columns firstDataRow firstDataColumn **Gz**

This shows the size of the *cellTable* in rows and columns (1 means 1 row or column). It also shows the row index and column index of the first row/column containing data (0 means first row). Only 1 row/column of labels is allowed, so *firstDataRow* and *firstDataColumn* can be only 0 or 1. The other parameters can be anything from 1 to whatever will fit in memory. There must be at least one row and one column in the table. This operator must be followed by however many **Gc** operators it takes to fill the table with data.

cellValue$_1$ *cellValue*$_2$ *cellValue*$_3$ … *cellValue*$_X$ **Gc**

This operator reads in cell values, one by one, and puts them in the table. Every cell in the table must be enumerated, even if it’s empty. Use as many invocations of **Gc** as necessary until the table is full or until all subsequent cells are empty. The first cell value goes to the top-left cell, subsequent cells along the top row are filled until the number of columns (as specified in the **Gz** operator) has been reached. Then the next row is filled from left to right.

The number of parameters to **Gc** doesn’t necessarily match the number of columns in the table; it is constrained by the following rules: only one parameter can be a string and there cannot be more than 15 parameters. This means that if the top row contains any labels, there will probably be many **Gc** invocations, each with one parameter; then for the data rows there will be fewer invocations, each with a greater number of parameters. String parameters can be any length up to 255 characters (the maximum line length for illustrator is 128 characters), and numerical values can be anything. Empty cells are denoted by a set of empty parentheses.

For example, the following table and **Gc** operator and its operands are equivalent:

	FIRSTCOLSECONDCOLTHIRDCOL	
GLOVES	131	
BALLS	225	
BATS	342	

```
%_() Gc (FIRSTCOL) Gc (SECONDCOL) Gc (THIRDCOL) Gc (GLOVES) Gc 1 3
  1 (BALLS) Gc 2 2 5 (BATS) Gc 3 4 2 Gc GC
```

col $width_1$ $width_2$ $width_3$ … $width_X$ *numParams* **Gw**

This overrides the default column width (expressed as a number of characters that fit in the table) given in the **Gy** operator. The first parameter is the column for which to start setting widths (0 is the first column); the next parameters (there must be at least one, and can be up to 14) set the width of that column and the next *n*(<14). To set the width for more than 14 columns, use multiple invocations of **Gw**. If –1 is the column width, it is considered a placeholder default column width. The operand *numParams* is the number of parameters to the operator, including the col parameter.

GC

The cell table is finished. This is an indication to Illustrator that the temporary state and data structures it has set up to read in the cell table can be disposed.

End of the Functional Specification

This ends the functional specification of the graph. All the operators shown so far should be written out with every line beginning with the PostScript comment characters **%_** (<percent> <underbar>).

10.5 Graph Customizations

After Illustrator has created the functional specification (and before it writes any of the graph objects), it tracks any edits that the user may have made to the graph in the form of changes to that functional spec. Such changes are called *customizations*.

Much like Paint Style parameters, Illustrator maintains a “running” customization—only the changes to the customization appear in the file. The **Gt** and **GT** operators bracket the customizations.

Illustrator writes out the **GX** operator to add the customization to the list for the graph.

If a **GX** operator appears *before* another operator with that parameter has appeared—anywhere within the entire set of customizations—Illustrator assumes that the parameter holds the initial value. For some parameters, the initial value is not one of the legal values. This means that the operator which contains this parameter must *always* appear somewhere in the list of customizations before the **GX** for a customization for which this parameter is necessary.

For example, if **Gx** appears in a customization that has been defined to apply to a column, but the **Gr** operator has not appeared, it is assumed that the target column is 0 (zero) since that is the initial value. That is, there is an implicit 0 Gl (zero gee-ell) at the beginning of the list of customizations. This applies to *all* operators. Note that some, like **Gi**, have a default value that is *illegal*. This means that before the **GX** operator appears on a customization that applies to an axis, there *must* be a **Gi** operator.

As another example, the bar design type parameter's initial value is 0, but that's not one of the legal values. Thus, before the **GX** appears for a bar-design customization, the operator containing the bar design type—**GD**—*must* appear). For each graph in a file, the value is reset to its initial state; that is, the "running" customization is not carried from graph to graph.

Note that some operators take parameters that do not have a meaning in the particular context; in this case, the parameters are used to set structure fields in the customizations, but they are ignored.

There can be fields that necessarily do not want a value; there is a special construct for this: **--** (<hyphen><hyphen>). This is used when a customization sets, for instance, the paint style of a group but the user did not fill in the color of the group. In the case where the group contains objects of various strokes—and the user changes the Paint Style without wanting to change the objects' strokes—Illustrator must create a Paint Style that does not change the stroke style but *does* change whatever the user wants changed.The -- parameter and the **w** operator are used as part of the construction of a "Set Paint Style" customization, to make sure that this customization will not reset the stroke width of a target that has previously had its width set by another "Set Paint Style" customization.

Customization Operators

version **Gt**

This marks the beginning of users' customizations to the graph.The current version is 2.

GT

This marks the end of the list of customizations.

target graphCustomization **Gx**

This sets up the main part of the customization, itemizing the target of the customization and the basic customization type. The parameters are:

target — is an index into all possible pieces of a graph. This is the object to which the customization is applied. For each type of target, other parameters to other operators will have to be filled in. For instance, if the target is one entire axis (9), the **Gi** operator should precede the **GX** operator, denoting which axis is the target. Initial value: 0. Possible targets are:

0 entire graph
1 all series, including legends
2 one series, including legends
3 one series but not its legend
4 one data bar/line/wedge
5 all data marks
6 one series' and its legend's marks
7 one series' marks, but not legend's
8 one data line segment's mark
9 one axis, including text, ticks, line
10 category axis' main line
11 one axis' set of major tick marks
12 one axis' single major tick mark
13 one axis' set of tick labels
14 one axis' single tick label

DRAFT

15 all legends’ text
16 one legend’s text
17 one numerical axis’ main line
18 one legend’s box or line, not mark
19 one legend’s mark
20 all labels along category axis
21 one label of category axis
22 entire “shadow” object
23 every tick of one axis
24 all minor (small) ticks of one axis
25 one minor (small) tick of one axis

graphCustomization — This parameter enumerates the type of customization. If this is an Illustrator Customization, it indicates the *illustratorCustomization* (see the **Gp** operator). Initial value: 0. The values are:

0 Illustrator Customization
1 Set Series’ Graph Style
2 Set Column (Bar) Design
3 Set Mark Design

illustratorCustomization **Gp**

For customizations that involve general illustrator operators (see list just below), this operator enumerates the type of customization. This is ignored if the *graphCustomization* parameter to the **Gx** operator indicates that this customization is a graph-specific customization. Initial value: 0. Values (for *illustratorCustomization*) are:

0 Move/Shear/Rotate/Scale
1 Set Paint Style
9 Send To Front/Back
10 Set Character Style
11 Set Layout Style

changeMethod **G+**

Customizations usually reset properties of objects regardless of the previous value of the property. A few, however, involve adding or subtracting something from the previous property. There is an Illustrator operator to add two points to the type size; to mimic this customization, the **G+** operator indicates that the customization's values are to be added to the current values, rather than replacing the values. Initial value: 0. Values (for *changeMethod*) are:

0 Reset to new value
1 Add new value to previous value

Note *Adding or subtracting values doesn't apply to the values* in the customization itself. *It affects* how *those values will apply to objects when Illustrator recalculates the graph.*

sendToFront **G1** (gee-one)

For the Send-to-Front and Send-to-Back customizations, the parameter denotes front or back (since the *illustratorCustomization* parameter of the **Gp** operator is the same for both operators). Initial value: 0. Values are:

0 Send to back
1 Send to front

doFill doStroke fillColorStyle strokeColorStyle isAMask **Gf**

For the Set Paint Style customization, these parameter denote the settings of the fill and stroke radio buttons that are not covered by the standard Paint Style operators. For those path objects not in graphs, the operator that finishes the path object denotes the values. Since this operator does not create a path object, but merely denotes its properties, there needs to be operators specifically for these values. The parameters are:

doFill — Used to denote that a non-empty fill style has been selected. 1 means that the customization creates a fill (of style *fillStyle*, below). Initial value: 0. Values: 0 or 1.

doStroke — Same as *doFill*, except it applies to the stroking. Initial value: 0.

fillStyle — Denotes the style of filling for this customization. Initial value: 0. The values are:

0 black (or white)
1 process
2 pattern
3 custom color
4 blend (applies only to Adobe Illustrator for Windows Version 4.0)

strokeStyle — Denotes the style of stroking for this customization. The values are the same as for *fillStyle*. Initial value: 0.

isAMask — This object has had the Mask box checked in the Paint Style dialog. 1 means the object is a mask. Initial value: 0. Values: 0 or 1.

column **Gl** (gee-ell)

For targets that are a series or a data point within a series, this is the column index in the table that corresponds to the series. Note that 0 (zero) means the first column that contains graphable numeric data; that is, if the first column consists of labels, then the second column is considered zero. Initial value: 0.

row **Gr**

For targets that are a series or a data point within a series, this is the row index in the table that corresponds to the index within the series indicated by the **Gl** (gee-ell) operator. Note that 0 (zero) means the first row that contains graphable numeric data; that is, if the first row is labels, then the second row is considered zero. Initial value: 0.

whichAxis **Gi**

For a customization that applies to an object inside an axis, this operator tells Illustrator *which* axis the object is inside.

whichAxis For targets that are part of an axis, this parameter denotes the axis (left, right, bottom or top). Initial value: 0.

The values are:

1 Bottom axis
2 Left axis
4 Right axis
8 Top axis (Adobe Illustrator Japanese Edition only)

a b c d h v generalGraphType reserved1 reserved2 **Gm**

For customizations that use matrices (the *graphCustomization* parameter in the **Gx** operator is Illustrator Customization, and the *illustratorCustomization* parameter in the **Gp** operator is Move/Shear/Rotate/Scale). The parameters are:

a, b, c, d, h, v These are the values for the matrix. They (the *h* and *v*) are in artwork-coordinates, and all are decimal values. Initial values: 0, 0, 0, 0, 0, 0.

generalGraphType Some customizations only look good for certain general types of graphs; that is, shearing of the bars in stacked-column graphs would not look good if applied to the data in pie charts, yet when applied to grouped-column graphs it achieves a good effect. This denotes the general type of graph to which the customization should be applied. This parameter will have meaning in the case where the user reads this file into Illustrator, then changes the graph type. Initial value: 0. The values are:

1 Grouped- and Stacked-column
2 Scatter and line graphs
3 Pie charts
4 Area graphs
5 All graphs

reserved1 Set to 0.

reserved2 Set to 0.

designName designType repeatPartialType rotateLegend **GD**

This operator sets up the parameters for bar design customizations (see the **Gp** operator).

designName — This is the name of the design. The design was set up, just like a pattern, in the prolog. Initial value: zero-length string.

designType — This denotes the type of the design. Initial value: 0. The values are:

6 Vertically-scaled design
7 Uniformly-scaled design
8 Repeating design
9 Sliding design

repeatPartialType — For repeating bar design customizations, this denotes whether to chop the design representing any partial values, or to scale it. Initial value: 0. The values are:

16 Chop partial values
17 Scale partial values

rotateLegend — When using bar designs, the user can have Illustrator automatically rotate the design in the legend box. This denotes whether to do it. 1 means rotate the design in the legend box. Initial value: 0. Values: 0 or 1.

repeatEachValue **Ge**

For customizations that set the bar design (see **Gp** operator), and for which the design type is a repeat design (see **GD** operator), this operator denotes the value for each repetition of the design. Initial value: 0.0. Values: any decimal number but zero.

tickValue **Gv**

For targets that are either one tick mark or one tick label, this is the numeric value corresponding to that tick mark. That is, if the user changes to red the tick line next to the 100 on the tick axis, this number tells Illustrator that any tick line on that axis that corresponds to the value 100 (no matter how many tick lines there are or what their values), is to be red. Initial value: 0.0. Values: any decimal number.

Note — *This is the only customization operator that appears after the* **Gx** *operator within one customization.*

GX

This finalizes the customization, which has been described by any of the previous values.

target column row whichAxis illustratorCustomization graphCustomization ignored changeMethod **Gg**

Note *This operator is not written by Illustrator; all the parameters are redundant with parameters of other, more efficient, operators. While Illustrator does currently read this in, and some files written by beta versions of Illustrator 3.x may have this operator in it, it is highly recommended that the other operators be used instead. Future versions may not read this operator.*

This sets up the main part of the customization, itemizing the target of the customization and the basic customization type. The parameters are:

target	See “Gx” operator.
column	See “Gl” operator.
row	See “Gr” operator.
whichAxis	See “Gi” operator.
illustratorCustomization	See “Gp” operator.
graphCustomization	See “Gx” operator.
ignored	Unused. Set to 0 (zero).
changeMethod	See “G+” operator.

10.6 Graph Objects

Graph objects make up the second part of the graph. When Illustrator loads a file containing a graph, it draws the objects as described by this section without referring to the functional specification (at will, a user can tell Illustrator to recalculate the graph based on the functional spec).

Graph objects are essentially standard Illustrator objects, use Illustrator operators for their construction, and are subject to the current paint, fill, and stroke styles and the rest of the cumulative settings of the graphics state. However, in order to allow efficient editing of graphs, Illustrator uses a particular hierarchical grouping organization in memory. This has two effects on the file.

- Each graph object is followed by the **Go** operator, which tells Illustrator what that object is—data point, legend, axis, and so forth. See section section 10.1” for an illustrated look at the various parts of a graph.

- Graph objects are grouped in a way representing their hierarchy in memory using the **u** and **U** operators (just as with other Illustrator grouped objects). Because this grouping represents a data structure, there may be in the file one or more "empty" groups of **u** followed immediately by **U**; there may even be nested empty groups.

Note *When constructing an Illustrator file external to Illustrator which contains a graph, you must make sure that the organization of the empty groups is followed accurately.*

Organization of Graph Objects

The following table shows the hierarchical organization of graph objects. Each group is bounded by the **u** and **U** operators.

- Left Axis group
- Right Axis group
- Legend group
- Category Axis group
- Data
  - Series 0
    - Legend box
    - data series 0
  - Series 1
    - Legend box
    - data series 1
  - Series *n*
    - Legend box
    - data series *n*
- Axis

Graph Object Operators

target column row whichAxis **Go**

After each object that represents a major part of the graph, Illustrator must identify it: which part of the graph did Illustrator just read in? These parameters specifiy that. This operator is not needed for any part of the graph that is not covered by the following parameters. Note that some of the parameters are not needed for several of the object types, although place-holders must be used.

target This identifies what the object represents. This identifies which objects correspond to the *target* parameter of the **Gg** (obsolete) and **Gx** operators. Some are deduced from their location in the file; others must be specifically identified. The objects that must be identified are:

1 all series, including legends
2 one sereis, including legends
4 one data bar/line/wedge
5 all data marks
6 one series' and its legends' marks
8 one data line segment's mark
9 one axis, including text, ticks, line
10 category axis' main line
15 all legend's text
20 all labels along category axis
22 entire "shadow" object

DRAFT

GS

The graph is finished: both the functional specification and the objects representing the graph have been written out. This is not unlike the U operator, which finishes a group.

11 Script Trailer

The syntax for the script trailer of an Adobe Illustrator document is as follows.

<document trailer> ::= %%Trailer
{<proc set termination>}*

<proc set termination> ::=
proc_set_name /terminate get exec

For each procedure set (resource) that was initialized in the script setup, the trailer invokes its terminate procedure in reverse order.

```
Adobe_Illustrator /terminate get exec
   Adobe_pattern /terminate get exec
   Adobe_customcolor /terminate get exec
   Adobe_cshow /terminate get exec
   Adobe_cmykcolor /terminate get exec
```

12 Creating Illustrator Documents

Other applications may create documents to be edited subsequently by Adobe Illustrator. Illustrator requires very few of the prolog, setup, and trailer components, however, these documents cannot be printed until they are opened and re-saved within Adobe Illustrator. Figure 4 shows a minimal document that is still acceptable by Adobe Illustrator.

Figure 4 . *A minimal document acceptable by Adobe Illustrator*

```
<minimal document> ::=
                    %!PS-Adobe-3.0
                    %%BoundingBox: <llx> <lly> <urx> <ury>
                    %%EndComments
                    %%EndProlog
                    %%BeginSetup
                    <font encoding>
                    <pattern defs>
                    %%EndSetup
                    {<object>}*
                    %%Trailer
                    %%EOF
```

Section section 13.2" describes the format for the document template information required by Adobe Illustrator on the Macintosh computer. When opening a document without this template information, Adobe Illustrator displays a dialog box that prompts the user for the location of the template for the file (if any). By supplying the resource information directly in your minimal documents, you can avoid this slight inconvenience.

13 Illustrator on the Macintosh

On the Macintosh computer, the resource fork of an Adobe Illustrator document contains several ancillary resources. The following sections describe these Macintosh-based resources.

13.1 PICT Resource

An Adobe Illustrator document may have a graphical screen representation provided so that a preview of the illustration may be manipulated on the screen by a page composition system. On the Macintosh, this representation is saved as a QuickDraw *PICT* picture resource within the resource fork of the document. The resource is given a resource type of *PICT* and a resource number of 256.

The picture's *picFrame* bounding box matches the bounding box of the illustration. This bounding box is specified in the **%%BoundingBox** comment in the prolog header. This is, the width and height of the *picFrame* are the same as the width and height, respectively, of the bounding box.

The picture resource is composed of two bitmap images: the image itself and its mask. If a particular bit is set in the mask, then the illustration has actually been painted the corresponding bit of the image. Otherwise, the corresponding bit has not been painted and should be considered transparent.

The mask is placed in the picture first, in the QuickDraw *srcBic* mode. It "punches a white hole" in just those areas that are painted. Then the image is placed in the QuickDraw *srcOr* mode, which fills in the punched areas, but leaves the other areas untouched.

13.2 TEMP Resource

This resource identifies the name of the document's template file, if it has one. The resource has three components (in order):

1. A 32-bit integer containing the directory identifier of the folder containing the template file. The integer is zero if the document has no template.

2. A Pascal string containing the name of the volume on which the template file resides. The string is empty if the document has no template.

3. A Pascal string containing the name of the template file itself. The string is empty if the document has no template.

The resource is given a type of *TEMP* and a resource number of 256.

13.3 PAGE Resource

This resource contains the *x* and *y* coordinates of the document's page origin as specified by the *Page* tool in Adobe Illustrator. The coordinates are in the default user coordinate system. In this system, the unit length along both axes is $1/72$ inch. The resource consists of two 32-bit fixed point numbers. The first specifies the *y* coordinate and the second specifies the *x* coordinate. The resource is given a type of *PAGE* and a resource number of 256

13.4 PREC Resource

This resource contains the standard 120 byte *Macintosh Printing Manager* print record. It describes the document's user-specified printing references as selected from the *Page Setup* and *Print* dialog boxes. The resource is given a type of *PREC* and a resource number of 256.

14 Save Options and Their Formats

Illustrator 3.x can save a file with or without a preview illustration and in several different ways. The last four methods write the same data fork to the file and vary in how and what they write to the resource fork.

- *No Preview omit EPSF header*. This option places no Preview in the file.
- *No Preview include EPSF header*. This option places no Preview in the file, but includes an EPSF header for the file.
- *B&W Macintosh PICT*. This option writes a Preview in the form of a black and white Macintosh PICT resource to the resource fork of the file.
- *Color Macintosh PICT*. This option writes a Preview in the form of a color Macintosh PICT resource to the resource fork of the file.
- *IBM PC (TIFF)*. This option writes a binary file with a pointer to TIFF data.

15 Adobe Illustrator for Windows, Version 4.x

Adobe Illustrator for Windows Version 4.x implements a feature generally called *Illustrator Format on the Clipboard* to facilitate cutting and pasting complex Bézier-based artwork and text effects between applications. The data for Illustrator Format on the Clipboard is actually a complete ASCII Adobe Illustrator file as defined in this document.

In Microsoft Windows, the ASCII clipboard format name used for Illustrator Format on the Clipboard is "ADOBE AI3." Programs which support Illustrator Format on the Clipboard should use this name in Microsoft Windows.

Currently, Adobe Illustrator for Windows Version 4.x supports cut, copy, and paste for Illustrator Format on the Clipboard. Adobe Streamline 3.0 for Windows and Adobe TypeAlign 2.0 for Windows support cut and copy of Illustrator Format on the Clipboard but do not support paste.

15.1 Header Changes Under Windows

Adobe Illustrator for Windows Version 4.x uses a subset of the full Adobe Illustrator header structure. Figure 5 lists the subset of comments used in the Windows version header.

Figure 5 . *Header contents of Adobe Illustrator for Windows Version 4.x*

```
%%Creator:  Adobe Illustrator(TM) version
   %%For:  (username) (organization)
   %%Title:  (illustration title)
   %%CreationDate:  (date) (time)
   %%DocumentProcSets:  Adobe_Illustrator_version level revision
   %%DocumentSuppliedProcSets:  Adobe_Illustrator_version level
   revision
   %%Documentfonts:  font…
   %%+font…
   %%BoundingBox: llx lly urx ury
   %%TemplateBox: llx lly urx ury
```

The principal difference for comments that convey arbitrary text information is how strings of characters are delimited. In documents of some versions of *Adobe Illustrator*, the individual strings are valid PostScript language strings; in other versions, the last text item in a comment is terminated by a newline character.

Adobe Illustrator for Windows Version 4.x requires four additional comments to provide information that is otherwise stored in the Macintosh resource fork by the Macintosh version of Adobe Illustrator.

%%Template: {*filename*}
%%PageOrigin: *x y*
%%PrinterName: {*printer brand name*}
%%PrinterRect: *llx lly urx ury*

DRAFT

%%Template: {filename}

The **%%Template** comment specifies the full pathname of the template for the illustration.

%%PageOrigin: x y

The **%%PageOrigin** comment specifies the coordinates of the document's page origin (in points) as specified by the *Page* tool. If you omit this comment, Illustrator sets the page origin to the ruler origin. See section 13.3" for more information about the page origin.

%%PrinterName: {printer brand name}

The **%%PrinterName** comment specifies the brand name of the printer for which the file is being generated—for example "Apple LaserWriter."

%%PrinterRect: llx lly urx ury

The **%%PrinterRect** comment specifies the bounding box of the image area of the printer identified in the **%%PrinterName** comment. *The coordinate system used by this comment is 4th quadrant rather than 1st quadrant as used in all other structuring comments.* This means that 0 on the y axis is at the upper left (rather than the lower left). If you omit this comment, Illustrator sets the default for letter size, portrait orientation as on the Apple LaserWriter. The Macintosh version of Adobe Illustrator ignores this comment.

15.2 Controlling the Grid in Windows and NeXT Versions

A grid allows objects to "snap to" an array of horizontal and vertical locations on the artwork.The capability of controlling the grid is part of Adobe Illustrator for Windows Version 4.x and Adobe Illustrator NeXT Version 3.x. Adobe Illustrator's **%AI3_Grid** comment appears immediately after the **%%EndResource** comment and immediately before the **%%EndProlog** comment.

%AI3_Grid.*version hzSpace vtSpace gridSnap R G B visibility*

The **%AI3_Grid** comment specifies the appearance of the grid and whether an object should snap to grid lines.

version — This is the version number of the grid specification. It is an integer. Note the decimal point. The current version number is 0.

hzSpace — This parameter is the horizontal spacing between grid lines in display pixels. The grid origin is at the page origin.

vtSpace — This is the vertical spacing between grid lines in display pixels.

gridSnap — This parameter controls the grid snap: enabled, horizontal, vertical, or both. The least significant bit specifies horizontal snapping; 0 means snap off, 1 means snap on. The next bit specifies vertical snap, and the next after that specifies whether the grid is enabled or disabled.

0 no snap specified; snapping disabled
1 horizontal snap specified, snapping disabled
2 vertical snap specified, snapping disabled
3 horizontal and vertical snap specified, snapping disabled
4 no snapping specified, snapping enabled
5 horizontal snap specified, snapping enabled
6 vertical snap specified, snapping enabled
7 horizontal and vertical snap specified, snapping enabled

R G B — Three parameters specifying the Red Green Blue color for the grid, in the range of 0 to 1.

visibility — This is an integer that encodes grid visibility and position. The least significant bit controls the grid position: 1 means draw grid in front, 0 means draw it in back of everything else. The next digit encodes grid visibility. This value is independent of the grid color.

0 invisible grid drawn in back
1 invisible grid drawn in front
2 visible grid drawn in back
3 visible grid drawn in front

16 List of Operators

A	locked	
b	close, fill and stroke path	
B	fill and stroke path	
c	curveto	
C	curveto	
d	setdash	
D	polarized fill style	
E	pattern	
f	close & fill path	(implicit **newpath**)
F	fill path	(implicit **newpath**)
g	fill setgray	
G	stroke setgray	
h	closepath	
H	closepath	
i	setflat	
I	path text	
j	**setlinejoin**	
J	**setlinecap**	
k	fill **setcmykcolor**	
K	stroke **setcmykcolor**	
l	**lineto**	
L	**lineto**	
m	**moveto**	
M	**setmiterlimit**	

n	path neither filled nor stroked	(implicit **newpath**)
N	close path neither filled nor stroked	(implicit **newpath**)
O	fill overprint	
p	fill pattern	
P	stroke pattern	
q	group (that contains clipping)	
Q	ungroup (from group that contains clipping	
R	stroke overprint	
s	stroke closed path	(implicit **newpath**)
S	stroke path	(implicit **newpath**)
u	group	
U	ungroup	
v	**curveto**	
V	**curveto**	
w	**setlinewidth**	
W	clip	
x	fill custom color	
X	stroke custom color	
y	**curveto**	
Y	**curveto**	
Z	text	
`	illustration	
~	illustration	
_	null	
@	pattern ink	
&	pattern path	

*	guide object
^	KanjiBitMapFont (Only Japanese version)
$	KanjiBitMapFont (Only Japanese version)
!	KanjiBitMapFont (Only Japanese version)
#	KanjiBitMapFont (Only Japanese version)
*u	compound path group
*U	compound path ungroup
*w	wraparound group
*W	wraparound ungroup

16.1 Text Operators

To	begin text object	type To – type = 0 -> point text 1 -> area text 2 -> path text
TO	end text object	– TO – also pops text matrix
Tp	begin text path	a b c d tx ty startPt Tp – startPt = start point value matrix = anchor matrix (concated onto CTM by TP)
TP	end text path	– TP –
Tm	set text matrix	a b c d tx ty Tm –
Td	translate text matrix	tx ty Td –
TM	pop text matrix	– TM –
TR	reset pattern matrix	a b c d tx ty TR –
Tr	set render mode	render Tr – sets the render mode: render = 0 -> fill text 1 -> stroke text 2 -> fill and stroke text

		3 -> invisible text 4 -> mask & fill text text 5 -> mask & stroke text 6 -> mask, fill & stroke 7 -> mask (only) text 8 -> filled text with stroked text following (patterned text only) 9 -> stroked text (preceded by render mode 8 text; patterned text only)
Te	end render	– Te –
Tf	set font name and size	fontname size Tf –
Ta	set alignment	alignment To – alignment= 0 -> left aligned 1 -> center aligned 2 -> right aligned 3 -> justified 4 -> justified including last line
Tl	set leading	leading paragraphLeading Tl –
Tt	set user tracking	userTracking Tt –
TW	set word spacing	minSpace optSpace maxSpace TW –
Tw	set computed word spacing	wordSpace Tw
TC	set character spacing	minSpace optSpace maxSpace TC –
Tc	set computed char spacing	charSpace Tc –
Ts	set super/subscripting (rise)	rise Ts –
Ti	set indentation	firstStartIndent otherStartIndent stopIndent Ti –
Tz	set horizontal scaling	scalePercent Tz –
TA	set pairwise kerning	autoKern TA – autoKern = 0 -> no pair kerning 1 -> automatic pair kerning
Tq	set hanging quotes	hangingQuotes Tq – hangingQuotes = 0 -> no hanging quotes 1 -> hanging quotes

TE	Set std platform encoding	(encoding pairs) TE –
TZ	Set encoding vector	(optional encoding pairs) newFontNameLiteral oldFontNameLiteral direction fontScript TZ –
Tx	non-justified text	textString Tx –
Tj	justified text	textString Tj –
TX	overflow text	textString TX –
TS	special chars	textString justified TS –
Tk	kern	autoKern kernValue Tk – autoKern = 0 -> manual kern 1 -> auto kern kernValue = kern value in em/1000 space
TK	non-printing kern	autoKern kernValue TK –
T*	translate matrix to start of new line	– T* –
T-	print a discretionary hyphen	- T– –
T+	discretionary hyphen	– T+ –

17 Document Syntax Summary

The notation <…>+ means one or more instances of the bracketed item. The notation {<…>}* means zero or more instances.

<document> ::=	<prolog> <script>
<prolog> ::=	%!PS-Adobe-3.0 EPSF-3.0 <header comments> %%EndComments {<proc set>}* (not required, normally present) %%EndProlog
<script> ::=	<setup> {<object>}* {<page trailer>}(not required, normally present) <document trailer> %%EOF
<setup> ::=	%%BeginSetup {%%IncludeFont: font}* {<proc set init>}* (not required, normally present <font encoding> <pattern defs> %%EndSetup
<pattern defs> ::=	{<pattern>}*
<pattern> ::=	%AI3_BeginPattern: (*patternname*) <E> %AI3_EndPattern
<page trailer> ::=	%%PageTrailer gsave annotatepage grestore showpage
<document trailer> ::=	%%Trailer {<proc set termination>}* (not required, normally present)
<object> ::=	{<A>}(object locking) <path object>| <path mask>| <composite object>| <text object>| <placed art object>| <subscriber object>| <graph object>| <PostScript document>

<path object> ::= <paint style>
<path geometry>
<path render>|<*> (<*> indicates guide operator)

<path mask> ::= <paint style>
<path geometry>
<h>|<H>
<W>
<path render>

<paint style> ::= {<color>|<overprint>|<path attributes>}*

<path attributes> ::= <d>|
<D>|
<i>|
<j>|
<J>|
<M>|
<W>|
%AI3_Note: <note>

<note< ::= up to 254 characters of arbitrary text.

<color> ::= <fill color>|<stroke color>

<fill color> ::= <g>|(fill black ink only, or)
<k>|(fill process ink, or)
<x>|(fill custom ink, or)
<p>(fill pattern)

<stroke color> ::= <G>|(stroke black ink only, or)
<K>|(stroke process ink, or)
<X>|(stroke custom ink, or)
<P>(stroke pattern)

<overprint> ::= <O>|(fill overprint, or)
<R>(stroke overprint)

<path geometry) ::= <m>
{<path operator>}*

<path operator> ::= <l>|<L>|<c>|<C>|<v>|<V>|<y>|<Y>

<path render> ::= <N>|(closepath; no fill no stroke)
<n>|(neither fill nor stroke)
<F>|(fill)
<f>|(closepath; fill)
<S>|(stroke)
<s>|(closepath; stroke)
<B>|(fill and stroke)
<b>(closepath; fill and stroke)

```
<composite object> ::=
                        <group object>|
                        <group with a mask>|
                        <compound path>|
                        <compound path mask>|
                        <wraparound group>

<group object> ::=      <u>
                        <object>+
                        <U>

<group with a mask> ::=
                        <q>
                        {<object>}*
                        {<masked object>}*
                        <Q>

<masked object> ::=     <mask>|<object>

<mask> ::=              <path mask>|<compound path mask>

<compound path> ::=     <*u>
                        <compound path element>+
                        <*U>

<compound path element> ::=
                        <path object>|<compound group>

<compound group> ::=
                        <u>
                        <compound path element>+
                        <U>

<compound path mask> ::=
                        <*u>
                        <compound path mask element>+
                        <*U>

<compound path mask element> ::=
                        <path mask>|<compound mask group>

<compound mask group> ::=
                        <compound mask bottom group>|
                        <compound mask non-bottom group>

<compound mask bottom group> ::=
                        {<A>}
                        <q>
                        <path mask>+
                        <Q>
```

```
<compound mask non-bottom group> ::=
                          {<A>}
                          <u>
                          <compound mask group>+
                          <U>

<wraparound group> ::=
                          <*w>
                          {<object>}*
                          {<wraparound objects>}*
                          <*W>

<wraparound objects> ::=
                          <text object>|<object>

<text object> ::=         <To>
                          <text at a point>|
                          <text area>|
                          <text along a path>
                          <TO>

<text at a point> ::=     <Tp>
                          <TP>
                          <text run>+

<text area> ::=           <text area element>+
                          {<overflow text>}

<text area element> ::=
                          <Tp>
                          <path object>
                          <TP>
                          <text run>+

<text along a path> ::=
                          <Tp>
                          <path object>
                          <TP>
                          <text run>+
                          {<overflow text>}*

<text run> ::=            {<text style>|
                          <paint style>|
                          <text positions>|
                          <Tk>}*
                          <text body>
```

<text style> ::=
<Tr>|(render mode)
<Tf>|(font & size)
<Ts>|(rise and fall)
<Tz>|(horizontal scaling)
<Tt>|(tracking)
<TA>|(automatic kerning)
<TC>|(intercharacter spacing)
<TW>|(interword spacing)
<Ti>|(indents)
<Ta>|(alignment)
<Tq>|(hanging quotations)
<Tl>(leading)

<text position> ::=
(printing only)
<Tc>|(computed intechar spacing)
<Tw>|(computed interword space)
<Tm>|(text matrix)
<Td>|(translate)
<T*>|(translate down)
<TR>(reset matrix; found only in pattern prototypes)

<text body> ::=
<Tx>|<Tj>|<T+>|<T–>

<overflow text> ::=
{<text style>|<paint style>|<TK>}*
<TX>|<T+>

<font> ::=
[
{<re-encoding pairs>}*
<Te>
{<re-encoding>}*

<re-encoding> ::=
%AI3_BeginEncoding
newFontName oldFontName
<TZ>
%AI3_EndEncoding <font type>

<font type> ::=
AdobeType|TrueType

<placed art object> ::=
<’>
<art reference>
<~>

<art reference> ::=
<file reference>|<file inline>

<file reference> ::=
%%IncludeFile: <filename>

<file inline> ::=
%%BeginDocument:<filename>
… included file contents
%%EndDocument

<filename> ::=
platform-specific path name of file

```
<subscriber object> ::=
                        %AI3_Subscriber:<subscriber ID>
                        <placed art object>

<subscriber ID> ::=     resource number of SECT resource in file

<graph object> ::=      <Gs>
                        <graph functional spec>
                        {<graph customizations>}
                        <graph group object>
                        <GS>

<graph functional spec> ::=
                        <graph size and dialog values>
                        {<graph subscriptions>
                        <graph axis>
                        <graph axis>
                        <graph axis>
                        <graph table specs>

<graph size and dialog values> ::=
                        <Gb>
                        <Gy>
                        <Gd>

<graph axis> ::=        <Ga>
                        <GA>

<graph table specs> ::=
                        {<Gw>}*
                        <Gz>
                        <Gc>+
                        <GC>

<graph customizations> ::=
                        <Gt>
                        {<graph customization>}*
                        <GT>

<graph customization> ::=
                        {<graph customization operator>}*
                        <GX>|<Gg>
                        {<Gv>}
```

<graph customization operator> ::=
{<Gm>}
{<Gf>}
{<Gy>}
{<GD>}
{<Ge>}
{<G1>}(gee-one)
{<Gi>}
{<Gl>}(gee-ell)
{<Gp>}
{<Gx>}
{Gr>}
{<G+>}
{<Gg>}
{<A>}
{<paint style>}*
{<text style>}

<graph subscriptions> ::=
<Gj>

<graph group object> ::=
<u>
{<graph rendered object>}*
<U>

<graph rendered object> ::=
<object>|<graph group object>
{<Go>}

<PostScript document> ::=
%%BeginDocument: <comment>
(arbitrary PostScript code)
%%EndDocument

<comment> ::= arbitrary text

DRAFT

Appendix: Changes Since Earlier Versions

Changes since May 4, 1991 version

- Updated entire document to include information on the Adobe Illustrator 3.x and 4.x file formats--DRAFT specification only.
- Reformatted in the new document format.

Changes since July 18, 1990 version

- The description for the operator **q** was changed from “except that the first object in the group specifies” to “except that some objects in the group specify.”
- The cover addresses were updated.

Changes since December 29, 1989 version

- The descriptions for the operators **o** and **a** in section 5.7 have been corrected. (They were reversed.)