POSTSCRIPT
LANGUAGE

TUTORIAL
and
COOKBOOK

ADOBE SYSTEMS

POSTSCRIPT LANGUAGE

TUTORIAL and COOKBOOK

ADOBE SYSTEMS
INCORPORATED

Addison-Wesley Publishing Company, Inc.
Reading, Massachusetts • Menlo Park, California
Don Mills, Ontario • Wokingham, England • Amsterdam
Sydney • Singapore • Tokyo • Mexico City
Bogotá • Santiago • San Juan

Library of Congress Cataloging in Publication Data

Main entry under title

Postscript language tutorial and cookbook

Includes index
1 PostScript (Computer program language)
I Adobe Systems
QA76 73 P67P68 1985 005 13'3 85-15694
ISBN 0-201-10179-3

Printed in the United States of America.
Published simultaneously in Canada

ABCDEFGHIJ-HA-898765
First printing August 1985

Preface

The POSTSCRIPT page description language provides a device independent standard for representing the printed page. This book is designed to be a companion piece to the *POSTSCRIPT Language Reference Manual*. It presents illustrative material to aid in understanding the POSTSCRIPT language. The tutorial information presented here has been deliberately separated from the reference manual to help ensure that the defining document offers a precise, unambiguous definition of the language and associated graphics imaging model. In all cases, when questions of definition or precise specification are raised, the *POSTSCRIPT Language Reference Manual* is the final word.

This book actually contains two documents: the *POSTSCRIPT Language Tutorial* and the *POSTSCRIPT Language Cookbook*.

The tutorial provides an easy, informal introduction to the POSTSCRIPT language and its graphics primitives. The tutorial's style and level of presentation is aimed at programmers who wish to design and implement applications, such as word processing packages, graphics illustrators, and CAD/CAM drawing systems. It is interactively oriented, and written with the assumption that you, the reader, already know how to program. You are encouraged to try variations of the examples presented in the tutorial on a POSTSCRIPT printer as you work your way through the book.

The cookbook is, as its name suggests, a collection of programs that are offered as examples of POSTSCRIPT usage. These samples have been chosen both as illustrations of the functional range of POSTSCRIPT and as useful ingredients for inclusion in application packages that you design. The cookbook samples demonstrate techniques for rendering quality graphics, achieving effective typography with digital fonts, and maintaining true device independence. Again, you are encouraged to experiment with variations of these samples on a POSTSCRIPT printer as you develop your own applications.

The principal authors of this material are Linda Gass and John Deubert. The final organization and the majority of the material for the *POSTSCRIPT Language Tutorial* is due to John Deubert. Ed Taft reviewed and proofread the material during the later stages of its production. Linda Gass designed and developed the *POSTSCRIPT Language Cookbook* and she is the principal author of both the examples and the explanatory text. The seminal idea of the cookbook is due to Doug Brotz and several of the illustrations in the cookbook are due to John Warnock. Andy Shore proofread the text and POSTSCRIPT sample programs. The book design was specified by Bob Ishi and was implemented by Andy Shore and Brian Reid. The index was compiled by Steven Sorensen.

The art of printing is rich in tradition, and the technology for producing the printed page has evolved over centuries. We at Adobe Systems are pleased to offer POSTSCRIPT as a tool for printing in the electronic age. I believe that this tutorial material will significantly enhance your ability to explore this exciting technology and help you enjoy the process of discovering the world of electronic printing.

Charles Geschke
August 1985

Contents

POSTSCRIPT LANGUAGE COOKBOOK

CHAPTER 1

INTRODUCTION

The POSTSCRIPT language is a programming language designed to convey a description of virtually any desired page to a printer. It possesses a wide range of graphic operators that may be combined in any manner. It contains variables and allows the combining of operators into more complex procedures and functions.

POSTSCRIPT page descriptions are programs to be run by an interpreter. POSTSCRIPT programs are usually generated by application programs running on other computers. However, many POSTSCRIPT printers, including the Apple LaserWriter, have an interactive state in which the user may program directly in POSTSCRIPT (see section 12.1).

1.1 POSTSCRIPT AS A PAGE DESCRIPTION LANGUAGE

POSTSCRIPT has a large selection of graphics operators that allow it to precisely describe a desired page. These operators control the placement of three types of graphics objects:

- **Text** in a wide variety of typefaces can be placed on a page in any position, orientation, and scale.
- **Geometric figures** can be constructed using POSTSCRIPT graphics operators. These describe the locations of straight lines and curves of any size, orientation, and width, as well as filled spaces of any size, shape, and color.

- **Sampled Images** of digitized photographs, free-hand sketches, or any other image may be placed on a page in any scale or orientation.

All graphic objects may be easily rotated, scaled, and clipped to a specified portion of the output page.

POSTSCRIPT Imaging Model

An *imaging model* is the set of rules that are incorporated into the design of a graphics system. The POSTSCRIPT imaging model is very similar to the model we instinctively adopt when we draw by hand.

The POSTSCRIPT model considers an image to be built up by placing ink on a page in selected areas. The ink may form letters, lines, filled shapes, or halftone representations of photographs. The ink itself may be black, white, colored, or any shade of gray. These elements may be cropped to a boundary of any shape as they are placed on the page. Once the page has been built up to the desired form, it may be printed on an output device.

Three concepts are central to the implementation of the POSTSCRIPT imaging model:

Current Page: The *current page* is the "ideal page" on which POSTSCRIPT draws. It is independent of the capabilities of the printer being used.

When a program begins, the current page is completely empty. POSTSCRIPT *painting operators* place marks on the current page, each of which completely obscures marks that they may overlay. Once the current page is completely described, it is sent to the printer, which reproduces the page as well as it can.

It is important to remember that no matter what color a mark has—white, gray, black, or color—it is put onto the current page as if it were applied with opaque paint.

Current Path: The *current path* is a set of connected and disconnected points, lines, and curves that together describe shapes and their positions. There is no restriction to the shapes that may be defined by the current path; they may be convex or concave,

even self-intersecting. The elements of the current path are specified in terms of their positions on the current page. The resolution of the printer in use in no way constrains the definition of the path.

The current path is not itself a mark on the current page. POSTSCRIPT *path operators* define the current path, but do not mark the page. Once a path has been defined, it can be stroked onto the current page (resulting in a line drawn along the path), filled (yielding solid regions of ink), or used as a clipping boundary.

Clipping Path: The *current clipping path* is the boundary of the area that may be drawn upon. Initially, the clipping path matches the printer's default paper size. The clipping path may be changed to any size and shape desired. If an imaging operator tries to mark the current page outside of the current clipping path, only those parts of the mark that fall within the clipping path will actually be drawn onto the current page.

Coordinate Systems

Positions on a page are described as x and y pairs in a coordinate system imposed on the page.

Every output device has a built-in coordinate system by which it addresses points on a page. We call this built-in coordinate system, idiosyncratic to each device, *device space*. Device space varies widely from printer to printer; there is no uniformity in the placement of coordinate origins or in horizontal and vertical scaling.

Positions on the POSTSCRIPT current page are described in terms of a *user coordinate system* or *user space*. This coordinate system is independent of the printer's device space. Coordinates in a POSTSCRIPT program are automatically transformed from user space into the printer's device space before printing the current page. User space thus provides a coordinate system within which a page may be described without regard for the particular machine on which the page is to be printed.

The POSTSCRIPT user space can be altered in three ways. The

coordinate system's origin may be *translated*, moved to any point in user space. The axes may be *rotated* to any orientation. The axes may be *scaled* to any degree desired, the scaling may be different in the x and y directions. A sophisticated user may specify any linear transformation from user space to device space. Thus, coordinates in a POSTSCRIPT program are changeable with respect to the current page, since they are described from within a coordinate system that may slide around, turn, shrink, or expand.

1.2 POSTSCRIPT AS A PROGRAMMING LANGUAGE

About one-third of the POSTSCRIPT language is devoted to graphics. The remainder makes up an entirely general computer programming language. The POSTSCRIPT language contains elements of many other programming languages, but most closely resembles the FORTH language.

POSTSCRIPT Stack

POSTSCRIPT reserves a piece of memory called a *stack* for the data with which it is working. The stack behaves like a stack of books. The last book placed on the stack is the first book that will later be removed. Similarly, numbers, strings, and other pieces of data placed on the stack will be removed in reverse order, the last item added to the stack being the first retrieved.

Postfix Notation

POSTSCRIPT operators that require numbers or other data, such as **add** and **sub**, retrieve that data from the stack. To use an operator, one must first place the data it requires, its *operands*, on the stack, and then call the operator. The operator will place its own results on the stack. This style of programming, in which the operands are specified before the operator, is referred to as *postfix notation*.

POSTSCRIPT Data Types

POSTSCRIPT supports many data types common to other languages, including reals, booleans, arrays, and strings. The POSTSCRIPT language also defines object types such as *dictionary* and *mark*. For descriptions of all the POSTSCRIPT data and object types, refer to the *POSTSCRIPT Language Reference Manual*.

POSTSCRIPT Flexibility

POSTSCRIPT is an extremely *flexible* language. Functions that do not exist, but which would be useful for an application, can be defined and then used like other POSTSCRIPT operators. Thus, POSTSCRIPT is not a fixed tool within whose limits an application must be written, but is an environment that can be changed to match the task at hand. Pieces of one page description can be used to compose other, more complicated pages. Such pieces can be used in their original form or translated, rotated, and scaled to form a myriad of new composite pages.

Printable Programs

POSTSCRIPT programs are written entirely in printable ASCII characters. This allows them to be handled as ordinary text files by the vast majority of communication and computer file systems. In addition, it ensures that a POSTSCRIPT program will be as easy for a person to read as the structure of the program allows.

CHAPTER 2

STACK AND ARITHMETIC

The POSTSCRIPT programming language, like all programming languages, works with various types of data, such as numbers, arrays, strings, and characters. The pieces of data manipulated by POSTSCRIPT are referred to as POSTSCRIPT *objects*.

There are many ways a language can manipulate data; for example, many languages require that data be placed in variables and be addressed by a variable name. The POSTSCRIPT language has variables, but it also manipulates data directly by using a special entity called a *stack*.

2.1 THE POSTSCRIPT STACK

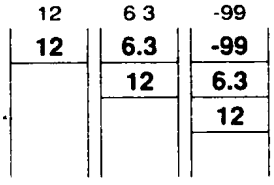
A stack is a piece of memory set aside for data which is to be immediately used by POSTSCRIPT. This memory area is organized in such a way that the last item put in is the first item available to be removed. This type of data structure is referred to as a *last in, first out* or *LIFO* stack.

A LIFO stack behaves like a stack of books. As the books are stacked up—Twain, then Dickens, then Hemingway, and so on—only the book on the top, the last one added, is really accessible.

Putting Numbers on the Stack

Any number appearing in a POSTSCRIPT source file (that is, a text file that contains a POSTSCRIPT program) is placed on the stack. For example, if a source file contains the following line

```
12 6 3 -99
```

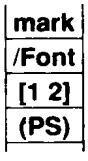


POSTSCRIPT Stack

the interpreter will take the following actions as it reads the line from left to right (see illustration at left):

1. Push the number *12* onto the stack
2. Place *6 3* on the stack, pushing *12* to the next position down
3. Put *–99* onto the stack, pushing the first two numbers down one place

The number *–99* is now at the top of the stack, waiting to be used. The other numbers are on the stack also, but can only be taken off in the proper order. It should be borne in mind as we use the stack that *any* kind of POSTSCRIPT object can be placed on the stack. This includes arrays, strings, and the more exotic POSTSCRIPT objects, like dictionaries. For the first chapter or two of this tutorial, we shall concentrate primarily on numbers, to simplify our discussion.



Anything can be placed on the stack

Note that spaces, tabs, and newline characters act as delimiters of POSTSCRIPT objects. Other characters, such as parentheses and brackets, can be delimiters under some circumstances; we shall discuss these as we progress through the tutorial.

2.2 ARITHMETIC

A POSTSCRIPT *operator* is a word that causes the POSTSCRIPT interpreter to carry out some action. It is the equivalent of the commands and procedures of other languages. When the interpreter comes across a word in a source file, it searches its internal dictionaries to see if that word is an operator name. If the name is listed in the dictionary, the interpreter carries out whatever instructions are associated with that name and then continues on to the next word in the source file. For more detail on POSTSCRIPT dictionaries, refer to chapter four.