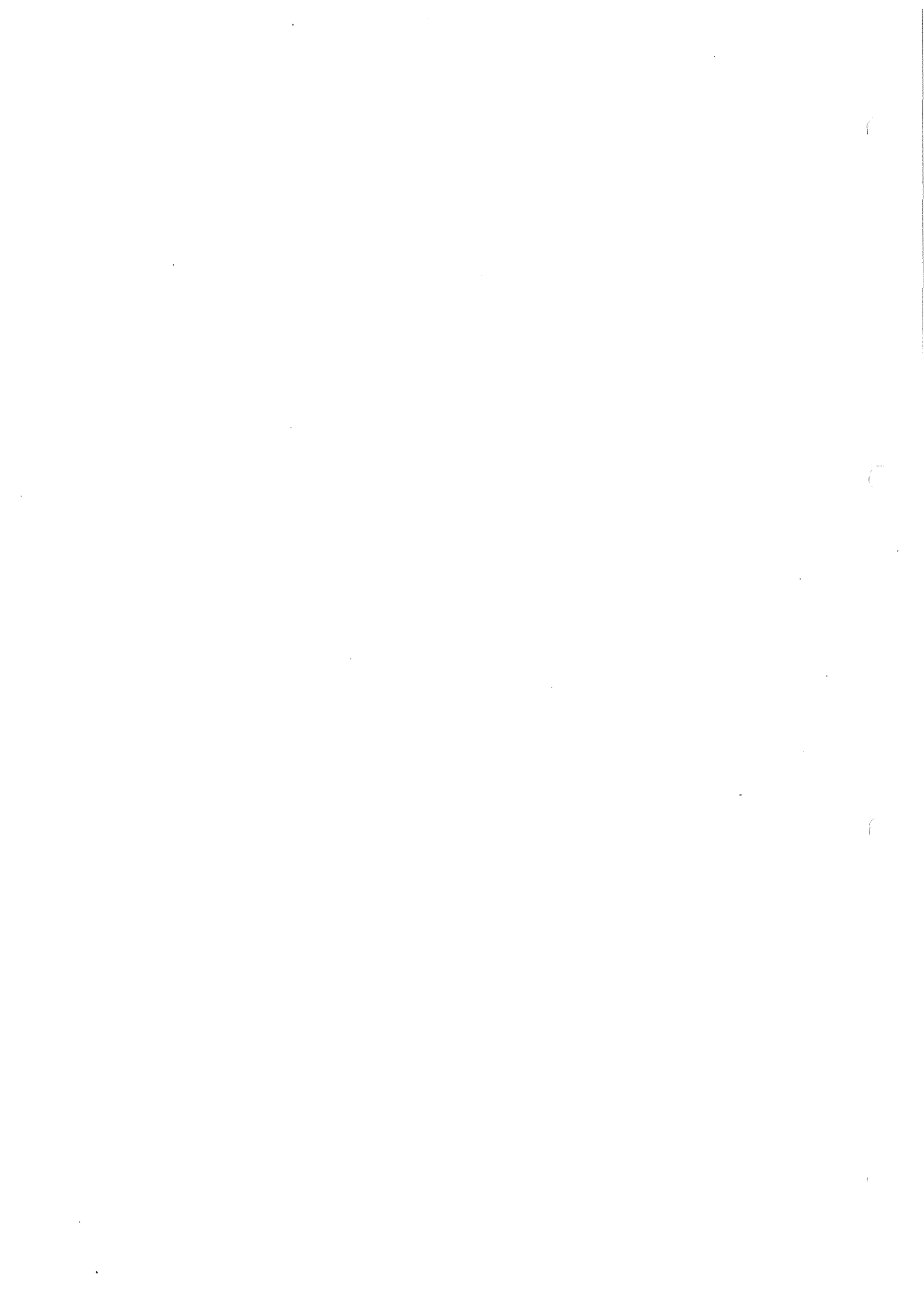


LaserWriter™ Reference







Contents

Figures and Tables x
Radio and television interference x

Chapter1	Introductionxx 1-1
	Original LaserWriter x 1-2
	LaserWriter Plus x 1-2
	The printing environment x 1-3
	Communication x 1-3
	Operating modes x 1-3
	Batch mode x 1-3
	Interactive mode x 1-4
	Emulation mode x 1-4
	Software environment x 1-4
	PostScript x 1-4
	QuickDraw x 1-5
	Font Manager x 1-5
	Printing Manager x 1-6
	LaserWriter driver x 1-7
	AppleTalk x 1-7
	An approach to your application x 1-7
Chapter2	Printing on the LaserWriter xx 2-1
	Working with the Printing Manager x 2-2
	The normal printing process x 2-2
	Choosing a printer x 2-3
	Setting up the page x 2-4
	Starting the print job x 2-5
	Using the print record x 2-5
	Managing the print job x 2-6
	Setting up for the print job x 2-6
	Initializing the software environment x 2-6
	Linking the printing code x 2-6

- Opening the Printing Manager x 2-7
- Validating the print record x 2-7
- Determining the selected printer x 2-7
- Optimizing your application x 2-8
 - Memory considerations x 2-8
 - Printable page size x 2-8
 - Clipping within text strings x 2-8
 - Protecting character image integrity x 2-9
 - Avoiding zero-width filled objects x 2-9
 - Tailoring your print loops x 2-9
 - Using the pPageFrame parameter x 2-10
 - Cancelling or pausing the print job x 2-10
 - Changing the local coordinate system x 2-10
 - Handling error messages x 2-11
- Working with the LaserWriter driver x 2-12
 - Using document control calls x 2-13
 - Using input/output control calls x 2-14
- Working around the Printing Manager x 2-17
 - Using PostScript through QuickDraw x 2-17
 - Working with text x 2-20
 - Working with polygons x 2-23
 - Rotating objects x 2-26
 - Printing forms x 2-26
 - Specifying PostScript commands x 2-27
- Modifying Printing Manager dialogs x 2-28
- Permanently modifying a printer driver x 2-32
 - Adding a sheet feeder to the LaserWriter x 2-34
- Designing a spooler for the LaserWriter x 2-36
 - Establishing dictionary status x 2-37
 - Obtaining the font list x 2-38
 - Coordinating non-coordinated fonts x 2-40
- Providing your own PostScript dictionary x 2-42
- Localizing the printing process x 2-43
 - Changing printer messages x 2-45
 - Changing Laser Prep messages x 2-45
- Using PostScript in a document x 2-46
- Intercepting PostScript files x 2-46

Chapter 3 Working with fonts xx 3-1

- Overview x 3-2
 - Understanding font terminology x 3-2
 - About screen fonts x 3-3
 - About printer fonts x 3-4
- Matching screen and printer fonts x 3-5

- Classifying fonts x 3-6
 - Style implementation x 3-6
 - Character set encoding x 3-7
 - Classifying fonts-release I x 3-8
- Style Mapping x 3-9
 - Encoding character sets x 3-10
 - Specifying printer font names x 3-10
 - Style mapping process x 3-11
- Naming fonts x 3-13
 - Release I font names x 3-13
 - Release II and later font names x 3-13
- Downloading fonts x 3-14

Chapter 4 Working in the printing environment xx 4-1

- Using AppleTalk x 4-7
 - Connecting a LaserWriter to AppleTalk x 4-7
 - Initializing the printer node x 4-7
 - Opening an AppleTalk connection x 4-7
 - Transferring data x 4-8
 - Closing the AppleTalk connection x 4-8
- Using PAP calls x 4-8
 - GetNextJob x 4-8
 - PAPOpen x 4-8
 - PAPRead x 4-10
 - PAPWrite x 4-11
 - PAPClose x 4-12
 - PAPUnload x 4-13
 - PAPStatus x 4-13
 - PAPRegName x 4-14
 - PAPRemName x 4-14
- Naming a LaserWriter x 4-14
- PAP packet formats x 4-14
- Detecting half-open connections x 4-15
- Accessing the LaserWriter directly x 4-16
 - Working interactively x 4-16
 - Working in batch mode x 4-17
- Using the Diablo 630 emulator x 4-20
 - Invoking the Diablo emulator x 4-20
 - Changing print parameters x 4-21
 - Deviation from Diablo protocol x 4-23
 - Detecting the end of a document x 4-23
 - Bold and double-strike x 4-23
 - Proportional fonts x 4-23
 - Paper positioning x 4-24
 - Unsupported commands x 4-24

MS-DOS communication parameters x 4-24

Figures and tables

Chapter1 ChapterTitle xx

- Figure1-1 The LaserWriter Printer xx 1-2.1
- Figure1-2 Macintosh/LaserWriter Printing Environment xx 1-4.1

Chapter2 Printing on the LaserWriter xx

- Figure2-1 Normal Print Process xx 2-3.1
- Figure2-2 Selecting a System Printer xx 2-4.1
- Figure2-3 Page Setup Dialog xx 2-5.1
- Figure2-4 Print Job Dialog xx 2-5.2
- Figure2-5 Font List Query xx 2-38.1
- Figure2-6 Font List Query Response xx 2-39.1
- Figure2-7 PostScript Code for Coordinating Fonts xx 2-40.1

- Table2-1 Printing Manager Errors xx 2-11
- Table2-2 Printing Manager Errors with the LaserWriter xx 2-11
- Table2-3 Document Control Calls xx 2-13
- Table2-4 I/O Control Calls xx 2-14
- Table2-5 Embedded Commands for PrintF and PrintR calls xx 2-15
- Table2-6 QuickDraw Comments xx 2-18
- Table2-7 Jump Table of Printing Manager Calls xx 2-33
- Table2-8 Localizable Resources xx 2-43

Chapter3 Working with Fonts xx

- Figure 3-1 style Mapping Table 3-9
- Figure 3-2 Char Set Encoding Table 3-10
- Figure 3-3 style Name Table 3-11.
- Table3-1 Font Terminology xx 3-2
- Table3-2 Font Classification—Release I xx 3-8
- Table3-3 Font Classification (Release II and Later) xx 3-9
- Table3-4 Macintosh Style Bits xx 3-12
- Table3-5 Style Code Format xx
- Table3-6 Temporary Font Resource Format xx

Chapter4 Working in the printing environment xx

- Figure4-1 AppleTalk Print Job Cycle xx 4-4.1

Figure4-2	PAP Packet Header Format xx 4-15.1
Table4-1	Operational Modes xx 4-3
Table4-2	PAP Call Summary xx 4-6
Table4-3	Persistent Parameters for Diablo Emulation
mode xx	4-21
Table4-4	eescratch Location Assignments xx 4-22

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for program coverage and details. In some countries the replacement period may be different; check with your authorized Apple dealer.

ALL IMPLIED WARRANTIES ON THE MEDIA AND MANUALS, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has tested the software and reviewed the documentation, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD "AS IS," AND YOU THE PURCHASER ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, even if advised of the possibility of such damages. In particular, Apple shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering such programs or data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

🍏 APPLE COMPUTER, INC.

This manual is copyrighted by Apple or by Apple's suppliers, with all rights reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of Apple Computer, Inc. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

© Apple Computer, Inc., 1986
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

Simultaneously published in the United States and Canada.

🍏 APPLE COMPUTER, INC.

Copyright © 1986 by Apple Computer, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc. Printed in the United States of America.

Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

Simultaneously published in the United States and Canada.

ISBN x-xxx-xxxxx-x

ABCDEFGHIJ-MU-xxxxxx

First printing, January 1986

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using the Apple Macintosh™ Plus and Microsoft® Word. Proof and final pages were created on the Apple LaserWriter™ Plus. POSTSCRIPT™, the LaserWriter's page-description language, was developed by Adobe Systems Incorporated.

Text type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Display type is ITC Avant Garde Gothic®. Bullets are ITC Zapf Dingbats®. Program listings are set in Apple Courier, a monospaced font.

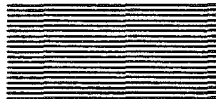
LaserWriter is a trademark of Apple Computer, Inc.

Macintosh is a trademark of McIntosh Laboratories, Inc., and is being used with express permission of its owner.

Microsoft is a registered trade-mark of Microsoft Corporation.

POSTSCRIPT is a trademark of Adobe Systems Incorporated.

ITC Garamond, ITC Avant Garde Gothic, and ITC Zapf Dingbats are registered trademarks of International Typeface Corporation.



Preface



About This Manual

Who Should Use It

This manual is intended for you if you want to do any of the following:

- Use PostScript to take full advantage of the LaserWriter's features in your application program.
- Access the LaserWriter from an Apple // or other computer. (You can connect the LaserWriter to any computer equipped with an RS232 or RS422 port.)
- Develop a font for the LaserWriter or the Macintosh.

LaserWriter Reference assumes that you are familiar with the information presented in the documents listed in the Related Documents section, below.

What It Covers

LaserWriter Reference describes the operating modes and special capabilities of the LaserWriter and LaserWriter Plus, as well as the hardware and software interfaces of these printers to the host computer.

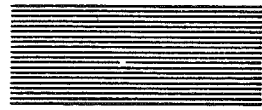
Conventions

Related Documents

Information pertinent to using the LaserWriter in a Macintosh environment can be found in the documents listed below.

- Item

- *LaserWriter and LaserWriter Plus User's Manual*. This manual explains how to set up a LaserWriter in the standard configuration as a network printer connected to AppleTalk, and gives basic operating information, such as how to load paper or change toner cartridges.
- *The PostScript Language Reference Manual*. This manual (by Adobe Systems, Inc.) describes the programming language that tells the LaserWriter what and how to print.
- *The PostScript Language Tutorial and Cookbook*. This manual (by Adobe Systems, Inc.) gives an introduction to PostScript and also provides a collection of useful PostScript programming examples.
- The "Printing Manager" chapter of *Inside Macintosh*. This chapter describes how to print from a Macintosh application using the generic Printing Manager/Printer Driver interfaces.
- The "Font Manager" chapter of *Inside Macintosh*. This chapter describes fonts, their structures, and how the Printing Manager communicates with QuickDraw.



Chapter 1



Introduction

The Apple LaserWriter™, shown in figure 1-1, is a laser-scanning, xerographic printer driven by a powerful microcomputer. The computer in the LaserWriter consists of a Motorola MC68000 microprocessor (the same microprocessor used in the Macintosh), one-half megabyte of read-only memory (ROM), and 1.5 megabytes of random-access memory (RAM). The ROM contains Adobe Systems' PostScript™ page description language, diagnostic code, AppleTalk® code, code to emulate the Diablo 630 printer, and a number of printer fonts. The RAM contains additional fonts and PostScript code downloaded by the printer driver, your application or the user.

Figure 1-1
The LaserWriter Printer

Original LaserWriter

The original LaserWriter has four font families resident in ROM:

- Times™
- Helvetica®
- Courier
- Symbol

❖ *Note:* The term LaserWriter is used in this manual to represent both the LaserWriter and the LaserWriter Plus. The terms *original LaserWriter* and *LaserWriter Plus* are used when it is necessary to distinguish between the two models.

LaserWriter Plus

The LaserWriter Plus increases the number of built-in fonts to eleven by adding the following seven:

- New Century Schoolbook™
- ITC Bookman®
- ITC Avant Garde®
- Helvetica Narrow
- Palatino®
- ITC Zapf Chancery®
- ITC Zapf Dingbats®

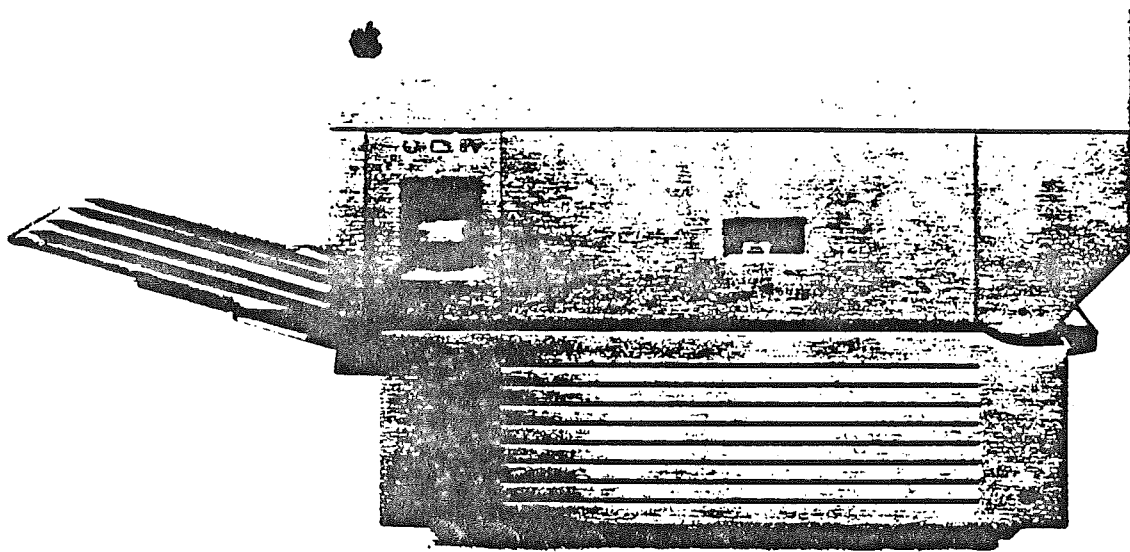


Figure 1-1. The LaserWriter Printer

pg 1-2.11

The driver for the LaserWriter Plus also supports downloadable fonts, thus improving both the quality and the speed of the printing process.

The printing environment

In normal operation, the LaserWriter continually cycles through the following three steps:

1. It begins by setting up a clean initial environment for executing the PostScript code that makes up a print job.
2. It obtains the job over a communication channel (see Communication, below), interpreting and executing the job as it receives it.
3. Finally, when it encounters an end-of-job or error condition, the LaserWriter cleans up its virtual memory, where the print image is constructed, and starts over at step one.

Communication

The LaserWriter is connected with the outside world via either AppleTalk or a point-to-point RS-232C or RS-422 serial link. (The four-position mode switch determines this choice.) This connection is referred to as the *communication channel* (or simply *channel*). A computer at the other end of the channel is referred to as the *host*. The host uses the channel to send PostScript programs and data to the LaserWriter. (The device at the other end of the channel can also be a terminal operated directly by a user, or a Macintosh running MacTerminal™.)

Operating modes

The LaserWriter operates in one of three modes: batch, interactive, or emulation.

Batch mode

Batch mode is the LaserWriter's normal operating mode, used almost exclusively for printing. In this mode, the LaserWriter accepts a job as a single PostScript program file. It executes that file

until it encounters an end-of-file or error condition. In this mode the LaserWriter transmits only the information specified by the executed code.

Interactive mode

Interactive mode is an option available to a user with a terminal connected directly to the LaserWriter. In this mode the user can experiment with PostScript to obtain various page description effects, or even use the LaserWriter as a host computer. PostScript is designed to be a general-purpose programming language, much like FORTH.

In interactive mode, the LaserWriter accepts input directly from the user. It echoes the input characters and provides limited editing capabilities. An interactive job terminates only when the user explicitly requests job termination.

Emulation mode

Emulation mode forces the LaserWriter to imitate another printer. In this manner programs written to print on other printers can (along with emulation code) print on the LaserWriter. While in this mode the LaserWriter expects incoming data to be combined text and control codes acceptable to the imitated printer.

The LaserWriter has code in ROM that enables it to emulate the popular Diablo 630 printer interface.

Software environment

Many programs within the Macintosh and the LaserWriter interact to print a Macintosh file on the LaserWriter. Figure 1-2 shows some of these programs and their interrelation.

Figure 1-2
Macintosh/LaserWriter Printing Environment

PostScript

PostScript is an industry-standard page-description language for describing text, graphics, and images for printed pages. It also can be used to control aspects of a printer's behavior. It is designed for

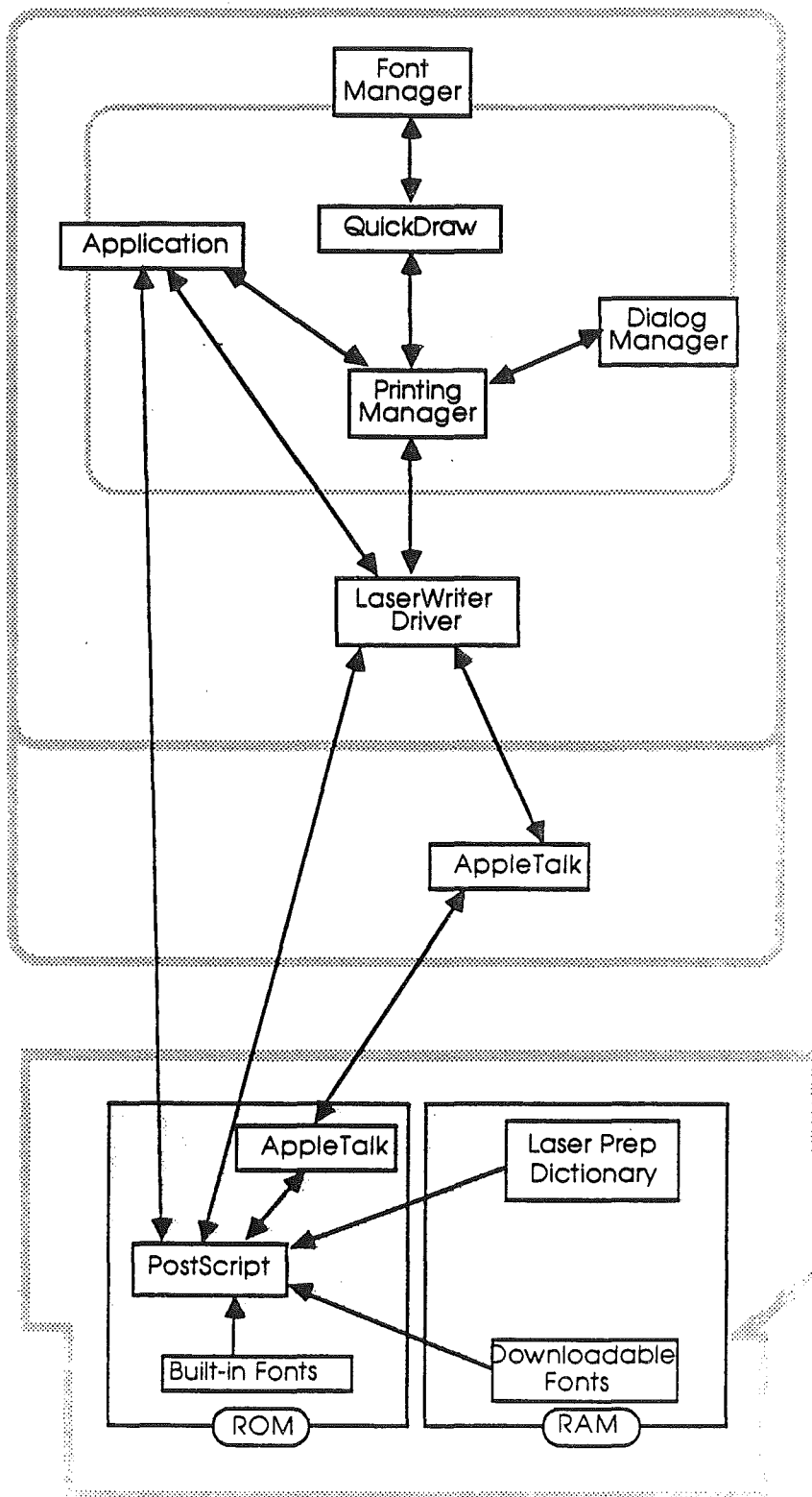


FIGURE 1-2. MACINTOSH/LASERWRITER PRINTING ENVIRONME

[pg 1-4.1]

use with high-resolution printers and typesetting equipment. The LaserWriter executes PostScript programs to construct the image to be printed.

Page description: PostScript programs are typically generated by application programs running on a host computer. For example, in normal printing with the Macintosh Printing Manager, the application uses QuickDraw commands to describe the image to be printed. The Printing Manager then causes these QuickDraw commands to be translated into PostScript programs in order to print on the LaserWriter.

The user, however, can write his own PostScript programs to achieve effects not available through Quickdraw, such as printing rotated text.

Printer control: Normally, PostScript programs are run to print pages. You (or the user) can, however, use PostScript to inquire about or to change values of some parameters in the LaserWriter itself (e.g., baud rate or parity settings), or perform some computation whose results are sent back over the communication channel.

❖ *Reference:* For more information about PostScript, see Appendix B, "Laser Prep Dictionary", and the Adobe Systems documents listed in the Preface.

QuickDraw

QuickDraw performs all Macintosh graphics operations (including text depiction), both on and off the screen. Using mathematical constructs, QuickDraw is able to create a wide variety of graphics. The QuickDraw interface is a set of general-purpose routines that can be accessed by your application, by the user, and by other parts of the Macintosh toolbox, such as the Printing Manager.

QuickDraw must call the Font Manager (see below) to obtain information about character sizes and styles before it can draw text.

Because the LaserWriter speaks PostScript, QuickDraw commands must be translated before the LaserWriter can execute them.

Font Manager

Whenever QuickDraw needs to draw text, it calls the Font Manager. The Font Manager controls specific information about how to draw the characters, including character widths, style implementation

algorithms, and kerning details. It also ensures that the most appropriate font definition is used in a given situation.

In the Macintosh/LaserWriter environment fonts can be divided into two basic categories: printer fonts and screen fonts. Screen fonts are defined to provide the best appearance on the Macintosh display screen. The screen, however, has a resolution of only 72 dots per inch, whereas the LaserWriter prints at the comparatively high resolution of 300 dots per inch. Consequently, printer fonts are usually defined differently than screen fonts.

❖ *Reference:* For more information about fonts and the Font Manager, see Chapter 3, "Working With Macintosh Fonts", and the "Font Manager" chapters of *Inside Macintosh*, volumes I and IV.

Printing manager

Due to the wide variety of current printing technologies, a general-purpose printer device driver would be either extremely limiting or impossibly large. For a system to be useful, however, it must provide a standard printing interface for a number of printers. The Printing Manager is that standard interface for printers connected to the Macintosh. It is a set of routines that allow an application to use QuickDraw to print on a printer without regard to device-specific details.

The Printing Manager performs the functions common to most print jobs; for example, it calls the Dialog Manager to allow the user to specify such parameters as page ranges and the number of copies.

Each printer is represented by a printer resource file which contains the printer driver. The driver translates communications between the Printing Manager and the printer, thereby providing device independence.

❖ *Reference:* For more information about the Printing Manager and printer drivers, see Chapter 2, "Working With the Printing Manager", and the "Printing Manager" chapter of *Inside Macintosh*, volume II

LaserWriter driver

The LaserWriter driver is a low-level Macintosh interface to the printer. It performs such functions as monitoring the print job status and ensuring that the necessary fonts are available on the printer.

The driver is usually called by the Printing Manager to perform printing and printer control. Your application can access the driver directly to alter the normal printing process.

AppleTalk

The AppleTalk network provides for efficient communication between the Macintosh and the LaserWriter. Some AppleTalk code is resident in both the Macintosh and the printer. Very few applications will be actually interact with AppleTalk, although everything an application sends to the LaserWriter must be passed through AppleTalk.

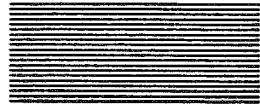
An approach to your application

If you are developing a Macintosh application, you should make every effort to utilize the Printing Manager whenever possible. The Printing Manager is designed to serve as a simple, general printer interface, enabling your application to use a wide variety of printer devices, and to avoid the pitfalls of device dependence. By using the standard interface, you help to ensure the longevity and usability of your product and to improve its upgrade path to new Macintosh hardware and software products.


However, the generality provided by the standard interface is achieved at a loss of some flexibility. To take full advantage of the LaserWriter's unique capabilities, you may have to modify or bypass parts of that interface. For example, you can alter or add to the standard printing dialogs, or even generate your own PostScript instructions to be passed to the LaserWriter. Such options are described in Chapter 2 for developers who require them.

If your application manipulates text to be printed on the LaserWriter, or if you are developing fonts for the LaserWriter, you should read Chapter 3.

You should read Chapter 4 if you intend to develop a print spooler on the AppleTalk network, if your application is designed to print to a Diablo 630 compatible printer, or if you are simply interested in the environment in which the LaserWriter performs its job



Chapter 2



Printing on the LaserWriter

The process of printing from a Macintosh to a LaserWriter can be surprisingly simple if your application does not need to print unusual text or graphics effects. In this case, using the standard printing interface enables you to write your application without concern as to what printer will be used: the user selects the printer via the Chooser application and specifies printing parameters, and the Printing Manager ensures that the best available method of printing is used.

This chapter describes the standard printing interface and how you can work with it (or work around it, if necessary) to allow your application as much control over the printing process as it needs.

Working with the Printing Manager

Selecting a print option from a menu to print a document on the LaserWriter can be deceptively easy. Several software packages interact on several levels to allow this ease. The Macintosh Printing Manager controls this interaction to remove the printing responsibility from your application, yet ensure that printing is performed properly.

The normal printing process

The printing process involves a set of intertwining events that occur before, during, and after the LaserWriter actually prints a page. Figure 2-1 shows how these events fit into the process.

MSC NNNN
ART: NN x NN
16.5 picas text to FN b/b

Figure 2-1
Normal Print Process

Choosing a printer

The printing process begins with the selection of a system printer. The user selects this printer by using the Chooser in combination with the Control Panel. For example, on an AppleTalk network with several LaserWriters and an ImageWriter, the user might see the dialog box in figure 2-2. By selecting the LaserWriter icon, he will see the names of the LaserWriters on the network. He can choose any of these as the system printer. This choice may be changed whenever printing is not being performed.

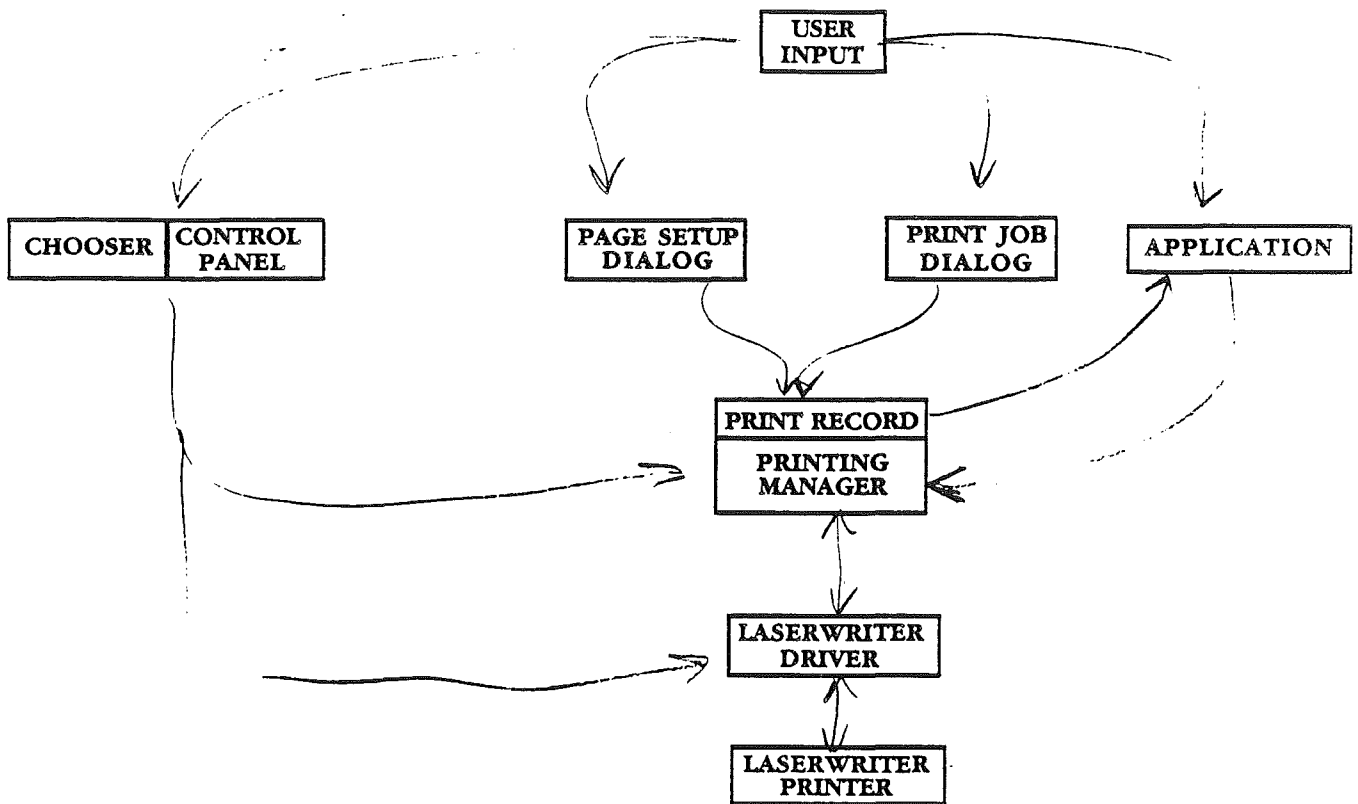


Figure 2-1. The Normal Printing Process

MSC NNNN
ART: NN x NN
16.5 picas text to FN b/b

Figure 2-2
Selecting a System Printer

❖ *Note:* The user can change the name of a LaserWriter by using the Namer program. Your application should not rely on the printer name remaining constant.

Selecting the system printer allows the Printing Manager to locate the necessary printer-dependent printing procedures (including the printer driver) and make them available to your application, so that you can make the same Printing Manager calls regardless of printer has been selected.

Setting up the page

Before printing begins, the user normally sets certain page characteristics by choosing the page setup menu option. You accept those characteristics by calling the *page setup dialog* or *style dialog* (via the Printing Manager call PrStDialog). This dialog allows specification of such printing aspects as paper size, page size, smoothing, font substitution, and orientation, as seen in figure 2-3.

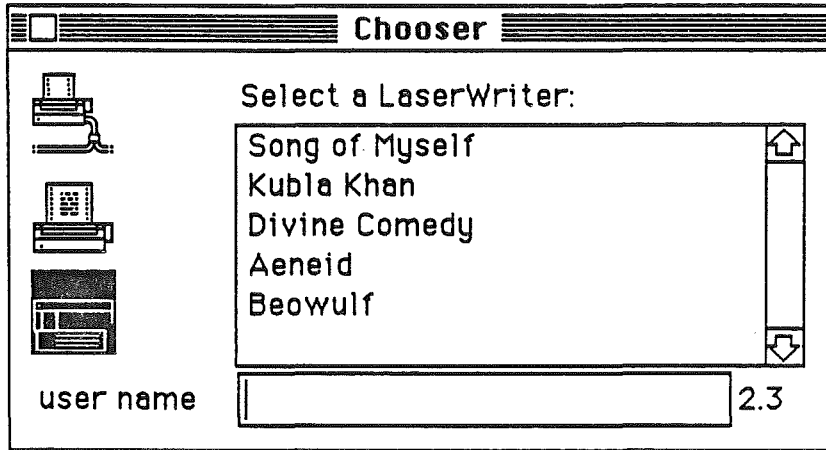


Fig. 2-4.1
Selecting a System Printer

Pg. 2-4.1

MSC NNNN
ART: NN x NN
16.5 picas text to FN b/b

Figure 2-3
Page Setup Dialog

Starting the print job

Printing on the LaserWriter is usually accomplished as a batch process known as the *print job*. The user initiates a print job by choosing a print option from a menu. At this time you provide the *print job dialog* (via the Printing Manager call PrJobDialog), asking him to specify options peculiar to the individual print job, such as the number of pages and copies to print. The standard print job dialog is shown in figure 2-4. Information obtained during the page setup and print job dialogs is stored in the print record for the job.



Using the print record

The print record contains information about the print job, such as the number of copies, printing style, and page dimensions. You should store this record in the document's resource file whenever you save the document.

LaserWriter v3.1

Paper: US Letter A4 Letter Reduce or Enlarge: %

US Legal B5 Letter

Orientation



Printer Effects:
 Font Substitution?
 Smoothing?

Margins:

Top: Left: Gutter:

Bottom: Right:

Fig 2-3. Page Setup Dialog

pg. 2-5.1

LaserWriter <Kubla Khan> v3.1

Copies: Pages: All From: To:

Cover Page: No First Page Last Page

Paper Source: Paper Cassette Manual Feed

Figure 2-4
Print Job Dialog

pg 2-5.2

❖ *Note:* The print record is used by the Printing Manager to create the proper print image. You should not change the print record directly.

Managing the print job

When the user clicks the OK button in the print job dialog, your application can use the necessary Printing Manager calls to formulate the image to be printed. The Printing Manager then supervises the printing process, including communication between the Macintosh and the LaserWriter.

Setting up for the print job

Setting up to print on the LaserWriter involves several steps on your part. Before you call the Printing Manager, you must first initialize the software environment. You can then open the Printing Manager and perform other steps such as determining the selected printer and validating the print record for the document.

Initializing the software environment

Before you call the Printing Manager, you must first initialize QuickDraw, the Font Manager, the Menu Manager, TextEdit, and the Dialog Manager. The appropriate initialization procedures (e.g., *InitGraf* and *InitFonts*) are detailed in *Inside Macintosh*. These procedures ensure that the necessary variables are initialized, the required heap space is allocated, and so forth.

Linking the printing code

In order to present a convenient printing interface, the Printing Manager appears to be a monolithic set of multipurpose routines. In reality, when you call the printer interface procedures, the Printing Manager dynamically links the appropriate executable printing code to your application, in accordance with the type of printer chosen by the user. To access the Printing Manager calls you must link your object code to *obj/PrLink.obj*. In your USES statement you should include the line `{ $U obj/MacPrint }` *MacPrint*. *MacPrint* is the application interface to the Printing Manager and the LaserWriter driver.

Opening the Printing Manager

When you have initialized the printing environment, you can open the Printing Manager with `PrOpen`, as explained in *Inside Macintosh*.

Validating the Print Record

The print record is stored with the document. Consequently, it can be mismatched with the current version of the Printing Manager or the currently selected printer. You should establish the validity of the print record when your application starts up and each time it interfaces with the print record (e.g., when it determines the size of the printable page). You validate the print record by calling the Printing Manager routine `PrValidate`; if the information in the print record is invalid for the current Printing Manager and selected printer, `PrValidate` corrects that information. The dialogs `PrStDialog` and `PrJobDialog` will call `PrValidate` when they are called. You should not call `PrValidate`, `PrStDialog`, or `PrJobDialog` between pages of a document.

❖ *Note:* You should not change the print record data directly. The Printing Manager needs this data to create the proper print image.

The format of the print record is described in detail in *Inside Macintosh*.

Determining the selected printer

Differences in printer capabilities sometimes make it important to know what type of printer is currently selected. The printers are designated as follows:

ImageWriter = 1

LaserWriter = 3

Your application can check the high byte of the `wDev` word in the `PrSt1` subrecord of the print record to determine the currently selected printer. The value in that byte will be a positive 1 or 3 depending on the type of printer currently selected. (Be sure to call `PrValidate` to ensure that you have the current print record.)

Optimizing your application

Constructing a page description is an intricate and time-consuming process. As you might expect, there are steps that you can take at the application level to speed up that process or make it more efficient. The following guidelines are presented to help you print a document quickly and efficiently on the LaserWriter.

Memory Considerations

On the LaserWriter you can print only in draft mode. (*Draft*, in this instance, refers to printing without spooling, rather than to printing a lower-quality copy.) Your data and printing code, therefore, will have to be resident in memory at the same time, requiring approximately 15K to 20K for the printer driver, AppleTalk, and other required code every time you print. If you are facing memory restrictions, you may want to bracket individual Printing Manager calls with `PrOpen` and `PrClose`. This approach closes the Printing Manager's resource file, eliminating the associated overhead.

Printable page size

It is important to make the distinction between the page size (the area of the paper that can actually be printed) and the paper size (the physical size of the paper) on the LaserWriter. The LaserWriter limits the page size in accordance with the paper size. To achieve uniformity across many different printers, the printing manager limits the page size even further: a non-printable border of 0.42" surrounds the printed page. Note that this is different from the print area available when using the Imagewriter. The printable rectangle is known as the page rectangle; its value is stored in the variable `prInfo.rPage` in the print record. The paper size is stored in the print record as the rectangle `rPaper`.

Clipping within text strings

The clipping region limits drawing to an arbitrary area within the `grafPort`. Although the drawing can be larger than the clip region, only the part within the clip region is actually drawn in the `grafPort`. Clipping text can be especially troublesome; each clipped character must be redrawn, and that takes time. Therefore, when

you need to clip strings, make sure that the clipping region or rectangle is larger than the bounding box of the text (i.e., be sure to clip only outside of the string bounds). Beware especially of ascending and descending parts of characters. Also, because of the difference between screen fonts and printer fonts, it can be difficult to clip to the correct characters.

Protecting character image integrity

One method of avoiding grafPort garbage is erasing before drawing. However, erasing before drawing text can cause the erasure of portions of characters on the previous line, particularly the descending portions of characters such as *g* and *y*; this can occur when an application uses an `eraseRect` whose height consists of the ascent plus the descent plus the leading.

You can avoid this problem by using a line spacing of the above height plus one pixel; this ensures the integrity of the descenders, but will make your line spacing wider. The alternative is to leave line spacing alone and make sure that your application does not use `eraseRect` calls.

Avoiding zero-width filled objects

QuickDraw objects that enclose zero pixels and are filled but not framed will not be displayed on the screen nor print on the Imagewriter, but they are real QuickDraw objects and will be drawn for printing on the LaserWriter. Even though these objects do not appear in the printed image, they do require time to print. You should avoid them whenever possible.

Tailoring your print loops

Most applications will not have to know on which printer they are currently printing. Because of disk space limitations, however, some applications spool a page and then print it when printing on the Imagewriter. Spooling is not currently performed when printing to the LaserWriter. In order to optimize for the LaserWriter though, you will probably want to have two sets of printing loops: one would spool a page and then print it (for the Imagewriter), and the other would print without spooling (for the LaserWriter). Since you can tell which printer is currently selected (see *Determining the selected*

printer), you will be able to switch correctly between the two methods.

◆ *Note:* LaserWriter printing will probably be accomplished through spooling on the 128K Macintosh, so be sure to call `PrPicFile` to print the spool file if `BJDocLoop` is set for spooling.

Using the `pPageFrame` parameter

The `pPageFrame` parameter is intended for scaling the `QuickDraw` picture of a page in a spooled file, for printing on the `ImageWriter`. When printing on the `LaserWriter`, this parameter is ignored and does not affect the print output.

Canceling or pausing the print job

If your application provides a procedure for handling the user's requests to cancel printing, with the option of pausing the printing process, beware of timeout problems when printing to the `LaserWriter`. Communication between the Macintosh and the `LaserWriter` must be maintained; if it is not, a no-response error will be generated and the `Printing Manager` will cancel the print job.

The `LaserWriter`'s default *wait timeout* period is thirty seconds. This period is the maximum time the `LaserWriter` will wait for input from the application before terminating the print job. This default cannot be changed via the standard `Printing Manager` interface. Your application should check whether printing is to be performed on the `LaserWriter`, in which case you should disable the pause/cancel option.

Changing the local coordinate system

When printing to the `LaserWriter`, the application will not be able to change the origin of the `portRect` for drawing purposes while in the `Printing Manager`'s printing loop (between the `PrOpenPage` and `PrClosePage` calls). A general workaround for this problem is to use `offsetRect`. Note that you can change the origin within the `PrOpenDoc` and `PrCloseDoc` calls; this facilitates printing multiple-page documents.

Handling error messages

If the Printing Manager gets an operating system error, it will put that error in low memory and terminate printing if necessary. Your application can retrieve such errors with a tcall to PrError. If an error occurs in the printing loop between PrOpenDoc and PrCloseDoc, the application should not jump out of the loop; it should continue through and let the Printing Manager terminate properly. For example, if an error occurs on PrOpenDoc, the application should ensure that PrCloseDoc is called. If an error occurs on PrOpenPage, be sure to call PrClosePage as well as PrCloseDoc.

Table 2-1 lists errors relevant specifically to the Printing Manager.

Table 2-1
Printing Manager Errors

Error Values	Constant	Description
0	noErr	No error
128	iPrAbort	Abort the printing process
-1	iPrSavePFil	Problem saving print file
-17		Unimplemented control instruction
-27	iIOAbort	Trouble with IO
-108	imemFullErr	Not enough heap space

In addition to the Printing Manager errors, several error messages apply specifically to the LaserWriter. Table 2-2 lists these errors.

Table 2-2
Printing Manager Errors With the LaserWriter

Error Values	Constant	Description
-4101		Printer closed or not found
-4100		Connection just closed
-4099		Write request too big
-4098		Request already active
-4097		Bad connection refNum
-4096		No free Connect Control Blocks (CCB's) available
-8150		No LaserWriter chosen.
-8151		Version mismatch between Laser Prep dictionaries.
-8159		No Laser Prep dictionary installed on LaserWriter.
-8160		Zoom scale factor outside legal range.

Error -8150 indicates that the user has not selected a system printer. This condition occurs only within the bounds of a PrOpenDoc and PrCloseDoc print loop. The Printing Manager handles this error, displaying an alert dialog prompting the user to use Chooser to select a LaserWriter on the network. It then clears the error condition before the end of the print loop.

Error -8151 indicates that the version of the Laser Prep dictionary in the printer does not match that of the Laser Prep dictionary in the system folder. This error also occurs only within the bounds of the print loop, and is handled with an appropriate alert dialog and cleared by the Printing Manager before the end of the print loop.

The Printing Manager automatically corrects both errors -8159 and -8160; these errors are not passed on to the application. Error -8159 occurs when printing is attempted before the LaserPrep dictionary is downloaded to the LaserWriter. Error -8160 occurs when the zoom scale factor—the percent of reduction or enlargement, usually specified in the page setup dialog—is out of range.

Working with the LaserWriter driver

If your application needs more direct control of the LaserWriter, you can bypass the standard interface of the Printing Manager, and issue calls directly to the low-level LaserWriter driver. The calls your application makes to driver routines are the same for all printers. However, printer drivers are device-dependent, and therefore must function differently.

Warning This approach is not as safe as using the standard Printing Manager interface. The usable life and range of product applicability of your application can become limited by the device-dependent nature of printer drivers.

The generic interface to printer drivers' control routines is a four-parameter control call:

```
PrCtlCall[iCtl:Integer; lParam1,lParam2,lParam3:Longint];
```

Using document control calls

You use document control calls to direct the LaserWriter in processing job initiation, execution, and termination. The document control calls, `iPrDevCtl [iCtl=7]`, match those of the Printing Manager. These calls take the form `PrCtlCall[iPrDevCtl, lParam1, 0, 0]` where `lParam1` is composed of two integers, `iHigh` and `iLow`. The document control calls for the LaserWriter driver are shown in table 2-x.

Table 2-3
Document control calls

Variant	iHigh	iLow	Meaning
DocOpen	1	iCopies	Open the printer and prepare to transmit data. iCopies is the number of copies of each page to make.
PageOpen	4	0	Initialize PostScript and driver buffers for a new page.
LineFeed	3	iAdvance	Advance the PostScript pen by iAdvance printer's points. If negative, use 1/6th inch.
PageClose	2	0	Flush buffers and send PostScript for printing iCopies pages.
DocClose	5	0	Close the printer connection and release the driver buffers.

The control calls formerly named `Reset` and `PageFeed` have been renamed `DocOpen` and `PageClose`.

Two new controls, `PageOpen` and `DocClose`, have been added.

❖ *Note:* The LaserWriter driver attempts to detect the omission of the control calls `PageOpen` and `DocClose` and to insert them where required, but you should not depend on this feature.

The `LineFeed` control obeys the convention that negative amounts signify a linefeed of 1/6th inch, zero signifies carriage return, and positive numbers are in bits; 1/72nd of an inch on LaserWriter.

Using input/output control calls

The driver input/output control calls use `iPrIOctl [iCtl=5]`. There are six control calls, each identified by a unique value in the low byte of `lParam3`. Table 2-4 shows these input/output control calls.

Table 2-4
I/O Control Calls

Variant	lParam1	lParam2	lParam3
ShowBuf	pBuf	iBytes	0
StdBuf	pBuf	+/- iBytes	1
HexBuf	pBuf	iOffset +/- iBytes	2
Fill	bFill	iLines iBytes	3
PrintF	pFmtStr	pArgs	4
PrintR	iResID/iIndex	pArgs	5

The `showBuf` call takes a pointer in `lParam1` and a length in the low integer of `lParam2` as text to be streamed to the printer. It does this by embedding the data in a PostScript show call.

The `stdBuf` call sends data to the printer without the PostScript show call. If the count is negative, the data is treated as PostScript text. This means that parentheses and the backslash characters are preceded by the PostScript backslash escape character, and that characters in the upper 128 range (i.e., with the most significant bit set) are sent as octal.

The `hexBuf` call takes a pointer in `lParam1`, a shift count in the high integer of `lParam2`, and a count in the low integer of `lParam2`. The data is sent to the printer as ASCII hexadecimal data. The `iOffset` quantity specifies the number of bits to skip over, providing a bit addressing capability. If the count is negative, the data is inverted before it is sent to the printer.

The `fill` call sends a single byte to the printer many times. It formats the data as `iLines` of data with `iBytes` of data on each line. Each line is separated by a carriage return. Thus the total amount of data is the product of `iLines` multiplied by `iBytes`.

The `printF` call is a formatting package, similar to the `printf` function in a 'C' library. `lParam1` points to a string with embedded commands. `lParam2` points to the data used by the commands.

The `printR` call is similar to the `printF` except that its format string is in a string index resource. The high integer of `lParam1` is the resource id, and the low integer is the index. The resource type is `POST=STR#`.

Table 2-5 shows the commands used in the `printF` and `printR` calls.

Table 2-5
Embedded commands for `printF` and `printR` calls

Command	Args	Meaning
<code>^i</code>	integer	The argument is converted to signed decimal ascii.
<code>^l</code>	long integer	The argument is converted to signed decimal ascii.
<code>^c</code>	char	The argument is a PostScript character. Parentheses and backslashes are preceded by an escape character, and characters greater than 127 are sent as octal.
<code>^b</code>	ptr, integer	The arguments specify a buffer and are treated exactly as in the <code>stdBuf</code> control call.
<code>^h</code>	ptr, 2 integers	The arguments compose a buffer and are treated exactly as in the <code>hexBuf</code> control call.
<code>^n</code>	-	A newline character is sent.
<code>^r</code>	long, 2 integers	The long is a resource type; the integers are a resource id/index pair. If the index is zero, the resource is treated as a string resource. If it is positive, it is treated as a string list resource. If it is negative, its size is determined by <code>getHandleSize</code> .
<code>^f</code>	integer	The argument is treated as a boolean.
<code>^^</code>	-	A single <code>^</code> character is sent.
<code>^R</code>	long, 2 integers	This command works the same as the <code>^r</code> command, except that the data immediately follows the <code>^R</code> in the format string itself.
<code>^s</code>	any	This command skips past any of the arguments used above but the quantum skip is exactly one integer or word size. It may be used to ignore certain arguments in an argument list so that the same argument list may be used for several different <code>printF</code> or <code>printR</code> calls.

<code>^p</code>	string pointer	The pointer points to a Pascal string whose length determines the number of bytes sent. This command works exactly the same as <code>^b</code> , but with no length argument required.
<code>^P</code>	-	This command works the same as <code>^p</code> , above, except that the integer length pointer follows the <code>^P</code> in the format string itself. Only absolute page zero pointers are valid.
<code>^e</code>	-	Causes a PostScript end-of-file. On AppleTalk, the PostScript session is closed, but the connection is not broken. On captured PostScript text files, a carriage return and control-D are recorded in the file, but it is not closed. Neither AppleTalk nor the text file are closed until printing is completed.
<code>^(</code>	-	Start PostScript text string conversion. This works just as a negative count in the <code>stdBuf</code> call described above, except that the negative count is not required. The string conversion continues until the <code>^)</code> command to end PostScript string conversion is encountered, (see below). Note that the left parenthesis is still sent as a data character, and the conversion is not begun until after the parenthesis is sent. This command cannot be nested.
<code>^)</code>	-	End PostScript text string conversion. This causes the effects of the Start PostScript text string conversion to be canceled (except that the negative count in <code>StdBuf</code> calls can still be used). The right parenthesis character is still sent as part of the data but the conversion ceases immediately before the parenthesis. This command cannot be nested.
<code>^t</code>	-	This command works just as <code>^r</code> except that the data in the target resource may also contain format escape sequences which are interpreted in the same manner as any other escape sequence. This provides a limited nesting feature. Note that any <code>^(</code> (and <code>^)</code> escapes are still in effect during the nesting.
<code>^T</code>	-	This command works just as <code>^R</code> except that the data in the target resource may also contain format escape sequences which are interpreted in the same manner as any other escape sequence. Note that any <code>^(</code> (and <code>^)</code> escapes are still in effect during the nesting.

Working around the Printing Manager

The standard printing interface provided by the Printing Manager is sufficient for most applications' printing needs. The information below is provided for those developers whose applications must provide features not accommodated by the Printing Manager.

Warning The longevity and product applicability of your application can be limited if it relies on features other than those provided by the standard Printing Manager interface.

Using PostScript through Quickdraw

With the QuickDraw picture comment facility, `picComment`, your application can take advantage of features that are available in PostScript but not in QuickDraw. For example, rotating text is not a feature of QuickDraw, but it is available through PostScript. Using `picComment` can also improve the efficiency of your application by performing certain operations (such as defining polygons) directly via PostScript rather than through the more general-purpose QuickDraw routines.

Six categories of picture comments are available through QuickDraw:

- text layout
- polygon definition
- line width definition
- object rotation
- forms printing
- PostScript command specification

Most of the picture comments are designed to be issued along with embedded QuickDraw commands that simulate (on the Macintosh screen) the commented commands. The LaserWriter driver processes picture comments, but other printer drivers (for example, the ImageWriter) do not. If the driver does process the picture comments, it ignores the accompanying Quickdraw simulated commands. If the printer driver does not process the picture comments, the QuickDraw commands are used. The comments and the embedded QuickDraw commands must occur in

the correct sequence to produce the correct state, as with any other QuickDraw facility.

The format of the picComment procedure call is as follows:

Procedure PicComment (kind, dataSize: INTEGER; dataHandle: Handle);

The kind parameter identifies the type of comment. DataSize is the size of that data in bytes. Datahandle is a handle to additional data (if appropriate).

Table 2-6 lists the picture comments recognized by the LaserWriter driver. A detailed description of each comment follows the table.

Table 2-6
QuickDraw Comments

Comment Type	Kind	Size	Data	Description
TextBegin	150	6	TTxtPicRec	Begin text function
TextEnd	151	0	NIL	End text function
StringBegin	152	0	NIL	Begin pieces of original string
StringEnd	153	0	NIL	End pieces of original string
TextCenter	154	8	TTxtCenter	Offset to center of rotation
LineLayoutOff*	155	0	NIL	Turn LaserWriter line layout off
LineLayoutOn*	156	0	NIL	Turn LaserWriter line layout on
PolyBegin	160	0	NIL	Begin special polygon
PolyEnd	161	0	NIL	End special polygon
picPlyByt	162	0	NIL	Byte size offsets
PolyIgnore	163	0	NIL	Ignore following poly data
PolySmooth	164	1	PolyVerb	Close, Fill, Frame
picPlyClo	165	0	NIL	Close the poly
SetLineWidth*	182	2	Point	Set fractional line widths
PostScriptBegin*	190	0	NIL	Set driver state to PostScript
PostScriptEnd*	191	0	NIL	Restore QuickDraw state
PostScriptHandle*	192	-	PSData	PostScript data in handle
PostScriptFile*	193	-	FileName	FileName in data handle
TextIsPostScript*	194	0	NIL	Send QuickDraw text as PostScript
ResourcePS*	195	8	Type/ID/Index	PostScript data in a resource

file

RotateBegin**	200	4	TRotation	Begin Rotated port
RotateEnd**	201	0	NIL	End Rotation
RotateCenter**	202	8	Center	Offset to center of rotation
FormsPrinting**	210	0	NIL	Don't clear print buffer after each page
EndFormsPrinting**	211	0	NIL	End forms printing after PrClosePage

* Implemented in LaserWriter driver version 3.0 and later.

** Implemented in LaserWriter driver version 3.1 and later.

As shown in the table, some comments are not implemented in the earlier versions of the LaserWriter driver.

Working with text

Line layout capabilities of the LaserWriter include justifying, rotating, or flipping lines of text, as well as spacing between lines. In keeping with the what-you-see-is-what-you-get philosophy, the LaserWriter performs extensive text line layout to maintain the format of the text as it appears on the Macintosh screen. Unfortunately, this feature can add as much as 50% to the required printing time—in excess of any time required by the application, should it do any text formatting of its own.

The Font Manager in the Macintosh Plus ROM has a feature known as fractional character width processing. This feature enables a screen font character to be defined in a one-point size as a single set of characteristics. The true width of the character can then be determined as a multiple of the one-point size. The Font Manager chapter of *Inside Macintosh, Volume IV*, contains an in-depth explanation of this feature.

The text layout comments (numbers 150 through 156) enable your application to optimize the LaserWriter's line layout performance, especially when used in conjunction with the true character width feature.

The text line layout algorithm is optimized to produce the best possible printer output across several point sizes, fonts, and styles. There are, however, cases wherein the position of a word or phrase can change within the line: the line layout function emphasizes good character and word spacing, and cannot guarantee that the position of an individual character or word will not change.

Formatting text: In order to specify formatting for text, you must be able to identify text that belongs together. You can use the comments `TextBegin` and `TextEnd` to bracket text that belongs in the same paragraph, and also to specify formatting for the text. After each `TextBegin`, you must use an associated `TextEnd` prior to any subsequent `TextBegin`: they are not designed to be nested. You can specify the following additional alignment, rotation, flip and spacing data in the `TextBegin` comment:

```
TYPE  
TTextPicRec = PACKED RECORD
```



```

tJus: Byte;    {justification - 0:none, 1:left, 2:right, 3:center,}
               { 4 or greater:full}
tFlip: Byte;   {coordinate flip - 0:none, 1:horizontal, 2:vertical}
tRot: Integer; {clockwise rotation in degrees - 0 to 360}
tLine: Byte;   {line spacing - 1:single, 2:1-1/2,3:double}
tCmnt: Byte;   {Reserved }
               END;

```

You use the `tJus` field to tell the driver whether your application will provide left, center, right, or full line justification. In these cases the printer driver requests the LaserWriter to measure each line of text being printed and to adjust its line length according to the Macintosh screen length. In the case of left, center, or right justification, the driver lets the printer's natural line lengths prevail unless they exceed the Macintosh screen length; the driver still must perform line length measuring to make this test. If your application does not use text layout comments, the driver always performs line layout as though you had specified full justification.

With the fractional character width feature, screen font and printer font character widths can be matched more precisely, so that line layout can be relaxed even more. When the printer driver detects that this feature is present, it will completely turn off its line layout algorithm if left, center, or right justification is specified. In this condition, no line length measuring is performed on the printer, allowing printing to proceed at its highest possible speed.

❖ *Note:* Your application can suspend execution of the line layout algorithm at any time by issuing the `LineLayoutOff` text comment (number 155). This comment forces suspension of line length measuring on the printer under all conditions, for all Macintoshes.

Reconstructing text strings: In order to avoid stack overflows, some applications draw long strings of text in short pieces. The LaserWriter needs a way to reassemble strings properly, without unusual gaps. You can use the `StringBegin` and `StringEnd` comments to bracket short strings of text, identifying them as sections of an original long string. You should perform any erasing or filling of background rectangles prior to the `StringBegin` comment so as not to disrupt the reconstruction of the original string.

Rotating text

QuickDraw does not have a facility for rotating text; text rotation has been the responsibility of the application. For example, MacDraw creates rotated text by intercepting `StdText` calls from `TextEdit`, drawing into an off-screen buffer, rotating the buffer, and performing a call to `CopyBits`. The result of the `CopyBits` call is a bitmap that can be displayed on the screen, or printed on the printer.

The printer may be required, however, to print a picture that was pasted from one application (MacDraw, for example) into another. The center of rotation used by MacDraw is the center of the bounding box for the complete object. When the picture is then cut and pasted into another application, the bounding box will have different coordinates in the application's `grafPort`. The printing code would have to reconstruct the object in order to calculate its new bounding box, calculate the center of the box, then rotate. Such an application-specific approach is extremely inefficient. Providing for a relative offset for communicating the center of rotation would significantly simplify rotation. This relative offset approach is provided by the `TextCenter` comment

When printing on the LaserWriter, text rotation is performed better by the printer itself. Resizing the bitmap of the rotated text can affect the integrity of the text image. Therefore, the original text data must be contained in the picture to make it available to the LaserWriter. To place the data into the picture, MacDraw actually draws the text in the picture, but first sets `clipRgn` to zero and makes the following calls before calling `copyBits`:

```
GetClip(saveClip);
ClipRect(zeroRect);
DrawString(textData);
SetClip(saveClip);
```

The `TextCenter` comment contains the offset from the present pen location to the center of rotation. The offset gives the y-component, then the x-component as `Fixed` numbers, allowing the center to be in the middle of a pixel. This comment should appear after the `TextBegin` comment and before the first following `StringBegin` comment. The data associated with the `TextCenter` comment is as follows:

```
TYPE
TTextCenter = PACKED RECORD
    yInt: Integer; {Integer part of y offset to center}
    { of rotation}
```

```

yFract:Integer; {Fractional part of y offset to center}
                {  of rotation}
xInt:Integer;   {Integer part of x offset to center of}
                { rotation}
xFract:Integer; {Fractional part of x offset to center}
                {  of rotation}
END;

```

Immediately following a `TextBegin` comment, the LaserWriter printer driver expects to see a `TextCenter` comment specifying the center of rotation for any text enclosed within the text comment calls. It ignores all further `CopyBits` calls, and prints all standard text calls in the rotation specified by the information in `TTxtPicRec`. The center of rotation is the offset from the beginning position of the first string following the `TextCenter` comment. The printer driver also expects the string locations to be in the unrotated coordinate system, i.e., in the coordinate system of the current QuickDraw port. The printer driver rotates the entire port to draw the text, so it can draw several strings with one rotation comment and one center comment. It is a good practice to enclose an entire paragraph or even several paragraphs of text in a single rotation comment so that the driver makes the fewest number of rotations at a time.

The printer driver can draw non-textual objects within the bounds of the text rotation comments, but it must reverse the rotation to draw the object, then re-rotate to draw the next string of text. To do this the printer driver must receive another `TextCenter` comment before each new rotation. Thus, rotated text and unrotated objects can be drawn inter-mixed within one `TextBegin/TextEnd` comment pair, but performance is slowed.

Rotated text comments are not associated with the orientation of the printer paper as selected by the Page Setup dialog. These are rotations with reference to the current QuickDraw port only.

◆ *Note:* All bitmaps and clip regions are ignored during text rotation so that clip regions can be used to clip out the strings on printers that can't take advantage of the comment capability. This has the unfortunate side effect of not allowing rotated text to be clipped.

Working with polygons

Polygons are defined in QuickDraw by specifying a sequence of line segments, each connected to the next at one common endpoint. The sequence of segments must be closed; that is, the last segment

ends where the first began. The line segment definitions within a QuickDraw polygon data structure are not directly accessible in Pascal, and are therefore not easy to manipulate.

Specifying a polygon: The polygon picture comments PolyBegin and PolyEnd bracket polygon line segments, providing an alternative way to specify a polygon, and allowing more control over its structure. All QuickDraw StdLine calls that occur between these two comments draw part of the polygon.

Closing a polygon: You can use the comment picPlyClo to specify that the current polygon should be closed. It is not sufficient to simply check that the starting point (begPt) of the polygon equals the ending point (endPt), to ensure that a polygon is really closed. This is especially critical for smooth curves because it can determine whether or not a sharp corner occurs in a curve. If you use this comment, you must use it immediately after a PolyBegin comment.

Using polygon comments with StdPoly: You can also use the polygon comments with the QuickDraw StdPoly call. If a FillRgn call is encountered before the PolyEnd comment, then the polygon is assumed to be filled. In order for a filled but not framed polygonal object to be deciphered from a QuickDraw picture drawn by MacDraw, MacDraw always frames its polygonal objects, using a pen size of zero width and zero height when no framing is otherwise needed.

One problem with the StdPoly call is that it assumes the data is already in integer format, which is not always true. MacDraw stores freehand objects as byte offsets, typically with several hundred nodes and no arbitrary upper limit. In documents that use 90% or more of memory, situations can easily arise where there is not enough memory to create a polygon that represents the freehand data, especially since the polygon, with integer data, requires twice as much memory. MacDraw polygonal objects are stored as fixed-point numbers. It takes more code to convert them to a QuickDraw polygon and then call FramePoly than it does to call LineTo directly.

Setting initial pen location: Unlike QuickDraw polygons, comment polygons do not require an initial MoveTo call within the scope of the polygon comment. This means that the polygon will be drawn starting from the current pen location at the time the polygon comment is received. You must therefore set the pen location before issuing a polygon comment

Smoothing polygons: A spline is a method used to fit a curve to the smallest number of points, or to determine the smallest number of points that define a curve. In MacDraw, splines are used to smooth polygons. The vertices of the underlying unsmoothed polygon are the control nodes for the quadratic B-spline (or Bézier curve) that is drawn. PostScript has a direct facility for cubic B-splines. The LaserWriter translates the quadratic B-spline nodes it receives from QuickDraw into the appropriate nodes for a cubic B-spline. You can use two comments specify smoothing for polygons: PolySmooth and PolyIgnore.

The PolySmooth comment specifies that the current polygon should be smoothed. This comment contains a byte of data as follows.

```

TYPE
PolyVerb =    PACKED RECORD
                f7:Boolean;           (not used)
                f6:Boolean;           (not used)
                f5:Boolean;           (not used)
                f4:Boolean;           (not used)
                f3:Boolean;           (not used)
                fPolyClose:Boolean;   (close the polygon)
                fPolyFill: Boolean;   (fill the polygon)
                fPolyFrame:Boolean;   (frame the polygon)
            END;
```

The PolyIgnore comment specifies that the remaining StdLine data is to be ignored. You can use it to define splines. When a smooth curve is to be put into the picture, the pen size is set to zero, and the defining nodes of the curve are drawn. Because the pen size is 0, these nodes are not displayed, but they are drawn. Then the pen size is set to the appropriate size and the smoothing algorithm is called, creating many short lines. The short lines are of no interest in re-creating the spline (and are even at the wrong resolution for printing on the LaserWriter), but the spline can be completely reconstructed from the defining nodes.

If framing is specified by the PolyVerb, the LaserWriter frames the smoothed polygon using the pen size at the time the PolyBegin comment is received. When the PolyIgnore comment is received, the LaserWriter ignores all further StdLine calls until the PolyEnd comment. For polygons that are to be smoothed, your application should set the initial pen width to zero after the PolyBegin comment so that the unsmoothed polygon will not be drawn by printers that cannot process polygon comments. To fill the polygon, the application should call StdRgn with the fill verb

and the appropriate pattern set, as well as specifying fill in the PolySmooth comment.

Rotating objects

The LaserWriter can rotate objects as well as text. Object rotation comments work much like the text rotation comments.

Starting the rotation: The RotateBegin comment (number 200) tells the driver that the following commands will be drawn in a rotated plane. This comment contains four bytes of additional data, a handle that points to the following data structure:

```
TYPE
Rotation = RECORD
    Flip:Integer;      {coordinate flip -0:none, 1:horizontal,}
                     { 2:vertical}
    Angle:Integer;    {clockwise rotation in degrees - 0-360}
END;
```

Stopping the rotation: The comment RotateEnd (number 201) is used to terminate a sequence of rotation comments.

Setting the center of rotation: The RotateCenter comment (number 202) specifies the relative center of rotation for objects just as the TextCenter comment does for text, but objects are drawn in the rotated coordinate system (specified by this comment) plus the pen location of the first object drawn in the rotated format. This comment must occur before the RotateBegin comment. The additional data for this comment is 8 bytes. The data structure of the accompanying handle is exactly like that of the TextCenter comment.

Printing forms

The form printing comments enable your application to modify information in fields of a form without forcing the LaserWriter to change or redraw the form.

Starting forms printing: You can use the FormsPrinting comment (number 210) to specify that the LaserWriter's print buffer is not cleared after each page is printed. As a result, when PrClosePage is called, the same data will be printed again, except that which has been whited out, modified, or erased by additional QuickDraw calls.

Stopping forms printing: The `EndFormsPrinting` comment (number 211) is used to stop forms printing after the next `PrClosePage` call. The print buffer is again cleared after each page is printed. This comment should be issued immediately before the last form page is printed.

Specifying PostScript commands

The PostScript comments notify the printer driver that the application will be communicating with the LaserWriter directly using PostScript commands rather than QuickDraw. Essentially, you use these QuickDraw comments to bypass QuickDraw itself. There are two ways to do so: your application can specify data in the comment handle itself, or it can point to another file which contains text to send to the printer.

Warning

You should use these comments very carefully. No error checking is performed on PostScript text that your application generates.

Setting the PostScript state: The `PostScriptBegin` comment (number 190) sets the driver state to prepare for the generation of PostScript commands by the application.

Restoring the QuickDraw state: The `PostScriptEnd` comment (number 191) restores the QuickDraw state of the printer driver.

Sending PostScript in text strings: The `PostScriptHandle` comment (number 192) points (via `PSData`) to the text of the PostScript commands to be sent. `PSData` is a generic handle that points to text, without a length byte, which is terminated by a return character.
xt

Sending PostScript in files: The `PostScriptFile` comment (number 193) tells the printer driver to send the PostScript commands contained in the file `FileName`. The format of this file is the same as that of a downloadable font file (see chapter 3).

Sending QuickDraw text as PostScript: The comment `TextIsPostScript` (number 194) specifies that all QuickDraw text following this comment is sent as PostScript. No error checking is performed. This comment is terminated by a `PostScriptEnd` comment.

Sending PostScript from a resource: The ResourcePs comment (number 195) causes the printer driver to send the PostScript commands contained in the specified resource.

Modifying Printing Manager dialogs

Because of the universality of the Printing Manager interface, flexibility is sometimes reduced. Occasionally a printer feature is needed by the application, but is not available through the normal interface. You may want to use unusual options or features of a printer or to supply printing parameters not available through the Printing Manager, but which fit logically into the print dialogs, particularly the Page Setup dialog. Your application can append (*piggyback*) its own dialog information to Printing Manager dialogs or supersede them with its own dialogs, rather than implement a discrete dialog which follows the Printing Manager dialog.

Appending an application dialog to a Printing Manager dialog is discouraged, because it almost guarantees obsolescence as the Printing Manager architecture changes with time. With the current architecture, however, there is occasionally not much of an alternative; therefore, the technique for modifying Printing Manager dialogs is described below.

The Printing Manager contains executable code that is essentially linked to the application dynamically when printer interface procedures are called. The correct printer procedures are determined in accordance with the current printer selection. Once these procedures have been linked, your application can intercept or patch into the printing dialog calls to append its own dialog interface. The application can add dialog options specific to itself or alter parameters already provided by the print dialog.

Warning

It is the responsibility of your application to perform patched operations with care and in conformance with normal print dialog activity. Even though an application may intercept dialog calls, normal print dialog procedures that have been patched must still be called in their proper sequence as part of the patch changes. The application must call the normal intercepted print procedures after it has performed its own operations.

You can change a dialog resource dynamically as long as you ensure that the resource is kept in conformance with the dialog resource format. You may choose to append to or alter information in both

the 'DLOG' and 'DITL' resource types. Rectangles may be modified without too much concern.

When you add items to a dialog, however, you must determine the current maximum number of items and add the additional number to that count. You must still pass all original dialog items to print routines to process, and process the new items with your own routines. The maximum number of dialog items must be determined dynamically, never programmed into your application; this number may change between printers and from one release to another.

The resource that contains the printing dialogs is contained in the currently selected printer driver file. To ensure that the current printer is activated, you can call `PrOpen`, which guarantees that the current printer driver resource file has been opened.

In addition to the familiar print manager dialog calls such as `PrStlDlg` and `PrJobDlg` there are three other calls available, `PrStlInit`, `PrJobInit`, and `PrDlgMain`. The Pascal entry syntax for these routines is as follows:

```
FUNCTION PrStlInit (hPrint:THPrint):TpPrDlg;  
FUNCTION PrJobInit (hPrint:THPrint):TpPrDlg;  
FUNCTION PrDlgMain(hPrint:THPrint; pDlgInit:ProcPtr):BOOLEAN;
```

`THPrint` is a handle to a valid print record as would be passed to either dialog during normal print procedure calls. `TpPrDlg` is a pointer to a print dialog which in reality is a normal dialog pointer with some print information appended.

The relevant portions of the `PrDlg` record are the dialog record portion itself followed by two procedure pointers, one for filtering keyboard input and the other for evaluating selected dialog items. These are referred to as `pFilterProc` and `pItemProc` respectively. (The first procedure pointer is offset in the `PrDlg` record by 170 bytes, \$AA hex, and the second pointer by 174, \$AE hex.) The pointer `pDlgInit` points to a Pascal procedure which `PrDlgMain` calls to initialize the dialog window. The calling syntax of this procedure is:

```
FUNCTION xDlgInit (hPrint:THPrint):TpPrDlg;
```

Appending your own information to a printer dialog involves several steps. First, your application must call `PrDlgMain` with a print record handle and a pointer to an initialization procedure. The initialization procedure is called by `PrDlgMain` to initialize the dialog appropriate for the selection made by the user. After the

initialization is performed, PrDlgMain shows the dialog and then loops, calling ModalDialog with pFilterProc and then pItemProc until the user terminates the dialog.

The initialization routine that you supply performs all the modification of the dialog. It reads in the dialog resources for the dialog, modifies the window rectangle to the necessary size (usually by increasing the bottom value), and then adds items to the item list in the form expected by the Dialog Manager's GetNewDialog function. (See the Dialog Manager chapter of *Inside Macintosh*.) This is the critical portion of appending information to a dialog; you must maintain the correct dialog format while adding to it.

Warning You must never rewrite a modified dialog resource to disk; if the Resource Manager procedures ChangedResource and UpdateResFile are called for the dialog resources you have modified, the changes will be permanent for all applications and will likely cause them to crash.

The xDlgInit procedure you supply to PrDlgMain must get the resource for the appropriate dialog. The resource ID numbers are \$E000 (-8192) for the style dialog and \$E001 (-8191) for the job dialog. The procedure must make its in-memory changes to these resources.

Next you call either PrJobInit or PrStlInit to actually create the dialog. This step must be performed no matter how completely or simply you perform the initialization. When this Print Manager initialization routine returns, you can modify the dialog record further by setting buttons, drawing initial conditions, or setting user procedures for some items.

Next you must note the entries that are in the pFilterProc and pItemProc locations of the dialog pointer record and store these for future reference. Then you can insert your own routines into these locations; your routines must internally call the procedures they replaced.

The pFilterProc is a normal number filter or keyboard entry filter procedure. You do not necessarily have to replace this procedure if it is not required. The procedure pItemProc is called with the following Pascal syntax:

```
PROCEDURE pItemProc (pDlg:TpPrDlg; itemHit:INTEGER);
```

where pDlg is the expected dialog record and itemHit is the item number that the user selected in the dialog window. This procedure must decide whether the item number is within the range of the

Printing Manager's normal hit list or is one of the appended items. If it is an appended item, your application handles it, without calling the Printing Manager's `pItemProc` procedure. If it is in the range for the Printing Manager, your application can take any further action it deems appropriate and then pass the item on to `pItemProc`.

If your application always calls the Printing Manager procedures where required, the display, item handling, and disposal of the dialog will be handled properly, the only real work required of the application is that of appending and handling its own items.

To call the additional printing procedures mentioned in the previous paragraphs, you need only declare the routines to be external in the syntax described and link your application with the print link module (`PrLink` or `PrintCalls`). In Pascal this can be done by the `EXTERNAL` declaration. In assembly language this can be done with the `REF` directive, and in MPW assembly with the `IMPORT` directive.

The following Pascal example shows a call to the style dialog after page setup has been selected from the File Menu.

```
VAR
    maximum      : INTEGER;
    ItemProc     : ProcPtr;
    FilterProc   : ProcPtr;

FUNCTION PrStlInit (hPrint:THPrint) :TpPrDlg;           EXTERNAL;

FUNCTION PrJobInit (hPrint:THPrint) :TpPrDlg;          EXTERNAL;

FUNCTION PrDlgMain (hPrint:THPrint; pDlgInit:ProcPtr) :BOOLEAN; EXTERNAL;

PROCEDURE MyStlInit (hPrint:THPrint) :TpPrDlg;
BEGIN
    hStlDLOG:=GetResource('DLOG',-8192);
    {Make changes to DLOG Resource}
    hStlDITL:=GetResource('DITL',-8192);
    maximum:=hStkDITL^(first integer in resource);
    {Make changes to DITL}
    PtrToDialog:=PrStlInit (hPrint);
    ItemProc:=PtrToDialog^.pItemProc;
    FilterProc:=PtrToDialog^.pFltrProc;           {Optional}
    {Make other calls such as SetDItem of DITL items or set certain
buttons}
    MyStlInit:=PtrToDialog;
END;
```

```

PROCEDURE pItemProc (pDlg:TpPrDlg; itemHit:INTEGER);
BEGIN
    IF (itemHit <= maximum) THEN ItemProc (pDlg,itemHit)
    ELSE {Do It Myself};
END;

```

```

FUNCTION MyStyleProcedure (hPrint:THPrint):BOOLEAN;
BEGIN {The main code to handle the style dialog}
    MyStyleProcedure:=PrDlgMain (hPrint,@pItemProc);
END;

```

Assembly language routines can also be called in a manner similar to the method above. Linking with the normal printer module, `PrintCalls` (formerly `PrLink`), will provide the necessary external procedure entry points.

Permanently modifying a printer driver

You can modify the printer driver in such a way that dialog modifications become permanently attached to the driver rather than to your application. In this case you must make the changes permanent by manually editing the printing manager resources or by writing the changes noted above back to disk as changed resources. In this way the driver becomes a new driver with differing features and characteristics from the original driver.

warning The process described below is very highly subject to obsolescence if new printer drivers and Printing Manager architectures are introduced in future machines and printers. The architecture described below is provided to assist those who must modify existing printer drivers. It is not guaranteed to remain the same in future releases.

❖ *Note:* If you change the driver, you should change the printer name (that used in both dialogs as well as that of the file), the signature, icons and device ID, so that it becomes a new print driver.

You must also patch or link procedures such as those above into the regular Printing Manager calls so that they become permanent procedure calls.

When you permanently attach such procedures to a printer driver, they can never share information through global variables. There are no global variables in Printing Manager routines. The globals

in the above example must be rewritten into the routines themselves or shared in some other way such as via a resource or a table compiled into the code.

Once you have written and compiled the procedures, they can be linked in a partial link process (essentially linked as a stand alone program with more than one entry point). Then they must be patched into the original Printing Manager code with additional resource editing operations. In the Printing Manager file is a resource, PDEF 4, which contains a jump table as the first 32 bytes of the resource. The format of this table is shown in table 2-7.

Table 2-7
Jump table of Printing Manager calls

Procedure	Byte Offset
-	0
PrStlDialog	4
PrJobDialog	8
PrStlInit	12
PrJobInit	16
PrDlgMain	20
-	24
-	28

Each entry in the table consists of a JMP instruction with an immediate offset into the rest of the resource. The jump table points to the procedures of the appropriate Printing Manager calls as described above. Once the routines are ready, you must append the code data to the PDEF 4 resource and displace the above table with a new table of similar structure pointing to the added routines. One way to accomplish this task is to simply precede the above table with a table containing the correct offset for the added routines. You can simply add 32 to the offset for procedures you do not modify.

To link from the added routines back into the original routines where necessary, you need only provide an assembly language glue routine which causes a jump to the beginning of the resource into the desired routine, offset by the correct amount to get to the original jump table.

Adding a sheet feeder to the LaserWriter

This section presents an example of how a special feature can be added to the LaserWriter driver—in particular, a sheet feeder. If you want to add a feature to the printer itself, it is necessary to also change the driver that drives the printer, since it must control, activate or otherwise monitor that feature. The example that follows is just one method of accomplishing this task.

A sheet feeder is attached to the printer itself, feeding sheets from its external trays into the external or manual feed slot of the LaserWriter when it is in operation. When a page is ready to be printed, the driver must decide whether or not the page must come from an internal tray, an external tray or manually fed by a human user (most sheet feeders still provide for this feature too). In this example the user chooses the correct feeder option tray decision in an expanded job dialog box (see Modifying Printing Manager dialogs, above).

The external sheet feeder is mechanically connected to the printer. The feeder is assumed to be activated or controlled via an RS232 cable connected, in this case, to the LaserWriter's 25-pin connector. This connector is a standard 9600 baud serial port which is not normally active when AppleTalk is active. The printer is controlled though AppleTalk, which in turn controls the sheet feeder through the 25-pin serial connector. A special piece of PostScript downloadable code must be invoked in order to use the 25-pin port during AppleTalk activity. (This piece of code is included below but must not be copied from the text. The code can be obtained on diskette from Apple.)

```
0 serverdict begin exitserver
currentfile eexec
178F14FF462224C4349FF73724059E7C6DC33B59279B12DAE8982C8724DC6E54
A10E69A6987227553F8D8E1B79DCDD44531DECDD4B49B845285B21537ABEE646
696C1A74A6CBF10611FFA2A6C0E2CB10D72B771E92E200899F41221FD9D34D30
53E451FBE903B1363B87411221BFC275C702AA24AB675972AE01C010638AA3CF
D7CDDE6CB5C758D1939D6ECC69599DD11DF039E4DBDC88995F5365F3B05574BA
3A99D0394918463C0FE790C5DBDDF67E1324BD1E8CFBE6FD60BE392D2E08E64A
D0FE29A2DE24A77D9B72ACD37920EA8921CA6D0B006DD6E00926BD2A2FBA1D97
10B0CF0A7ED55B23351FAEBD5A791A86A8064D3989851248382BA8383E1170EC
D3924BDE2CF829AA4FFE57A73625F230E4E27CC1FB5274E75A675D66870F4D94
54DF9A4BBEBE75240860808822C094FF3BDF83669B4831BE7C258E6C3C63BA4B
A3640ECBF745C94599BB92DF925D6D69A6ACE63BC8DFDDBEAF9606D407BCA123
EAFDAB921678642F50DC9F97EC35E45A19361EDEBBD408753B5D10DC4FB03BEA
68E750837CE20940F7226C9B436CF24120F8AB0F6A8C86F3A26D3646B69DB850
2FB86EF78832EB6D69ADA83576E8C968F857D0453936D7CB8A98F39985AC249D
3DD24D0BD7D96CCFFD8EA04DE6B3E94D7F4751206418A053B06674397CEA5464
```

```
157012E4D448E2F85B562432A84E11BF99AFDDCDAA0D90BB3377A92773703CD8
1BF8D71A841F1DF0F80757ACF06EF343449180D8FCBA93D605D048552826CD1A
81F07E7809BDA11E3840DE0D97807BBDFFD2D36B97EF469445A24B5B3924A561E
EE0A948B6B6B277BF7C3B498CD464C98F2A28FCAA3A2E94F404BE7080DE3CE07
AE536ADA2858B76FAB81F5FFAE27C168B8E100BAA7BE0BB7374EF31A9B2B740E
DB6EB926D2E7E7C4460BE8AB8ABC461EBE36BA7D52B3E09B91F2103B59FC6D02
36F4781F48E77FA5AF30A3F44E7990CA9754D595951FF1B17A4B0F9263305FB0
25596C3CC61D84CCAF481C538337991E09CDEE2B42FC623045C02D316F96A116
B44689008870A380D4FFDA5A44DB5E77428F084880C96EDFABCB8D3D3885EF7F
4E93768A28F48707A6446201248DB6807DD0359AD2D206894078EB52F16918AB
26E1B9C477C0BC1E27A2AF633EA2573CFA3BB6B1F6B9B2E45776EAE8CBF5ACF9
10053D5C8F5CCDAD45C6972D8BAE082A6192BEE42434AD1D8DE3577574BD55BC
009BA9DA0DD409E418DF24B2B14A48E5E2B60576278D9076A7832471C0EA922E
1B3F1A3E3923AC67A04ADAEFF70FF3842A7B1E0F7E91C096A25EA0F8CAB4FA7BDA
```

The above code implements several PostScript calls:

```
9600 3 setupRS232 % is used to setup the 25 pin connector and must be %
called at the beginning of every document.

nn writeRS232 % outputs a character to the 25 pin port where the %
character ordinal value (nn) is given in decimal.

readRS232 % returns (nn true) or (false) where false means no %
character available and true means nn is character
```

These PostScript calls must be polled calls; they are not interrupt-driven. This limits the speed at which I/O can take place, but is probably adequate for operating a general sheet feeder.

Other PostScript routines that the developer might provide are procedures to replace cypage and showpage that will cause a sheet to be fed from the desired paper tray or an appropriate internal tray as necessary. Part of this action might be to switch from internal feeder operation to manual feed (see the PostScript manual), send the appropriate codes to feed a sheet into the manual feed slot, and synchronize with the showpage call. The PostScript code for this process is dependent on the particular sheet feeder being supported.

When all the items described above are available for packaging the following steps should be performed:

- Assemble all the PostScript code into a single resource or set of resources as described in *Providing your own PostScript dictionary*, below. Special commands which are intended to handle tray selection, showpage/copypage replacement, operation of the 25 pin RS232 connector, and other special purpose activities can go in these resources. When the Printing Manager code is activated during the printing of a document, this PostScript code will be automatically downloaded for you.
- Develop your own dialogs to attach or append to the existing LaserWriter driver dialogs. This is described in *modifying Printing Manager dialogs*, above. In these dialogs you can provide tray selection buttons, tray sequence selections, or other options to the user.
- During printing, when your application determines that an external sheet feeder must be activated and controlled, it should send PostScript code in QuickDraw picComment commands through the Printing Manager as described in *Using PostScript through QuickDraw*, above. The showpage call will be made by the Printing Manager, so it is never necessary to do this yourself.
- If your product is a driver or sheet feeder, you will want to supply these capabilities to all applications; this is more difficult to accomplish. Selected options from the user cannot be easily passed through to the PostScript procedures provided above, since access to QuickDraw picComment calls is available only to applications. In this case it is still possible to provide sheet feeder operation by replacing often used PostScript operators with routines which perform the task automatically. It is also possible to provide dynamically modified PostScript routines, which are created each time a dialog session is completed, to reflect the options selected by the client. This PostScript can be the downloaded code mentioned above that is sent to the printer at the beginning of every document.
- ◆ *Note:* The process described in the above steps is perhaps awkward but is operationally sound. These techniques are by no means unique nor necessarily recommended by Apple but are intended to serve only as an example.

Designing a spooler for the LaserWriter

Currently, documents to be printed on the LaserWriter are not spooled. Consequently, during most of the time required to print a document, the user is unable to use the Macintosh. Spooling files to the LaserWriter for printing can significantly improve the

productivity of the network. A spooler effectively inserts itself in the network between the LaserWriter driver and the LaserWriter itself.

To implement a remote AppleTalk-connected spooler for the LaserWriter driver on Macintosh, you must be familiar with printer drivers. You must also understand the protocol used at the PostScript level in order to preserve this interchange in a spooling device. There are three main areas of concern when splicing into this interchange: establishing PostScript dictionary status, obtaining the font list, and downloading fonts the required fonts.

At the beginning of every print job on the LaserWriter, the LaserWriter driver interrogates the printer twice, in PostScript; first for the status of the Apple Laser Prep dictionary, and then for font information.

Establishing dictionary status

The first query, the dictionary status query, asks for information about the state of the Apple PostScript dictionary. It is preceded by the comment string, "%?appledict version #n.n", and is followed by the string, "%?end". The driver expects a single character response: 0 (not defined), 1 (dictionary present and correct) or 2 (dictionary present, but does not match version number sent). The n.n sequence above is the current version number that the driver knows about, in ASCII fixed point decimal notation, and which the printer is expected to verify for a correct match. Version numbers match (for the purposes of this query) if the integer part of the version number of the downloaded dictionary matches the integer part of the version number in the Apple Laser Prep dictionary. Higher fractional numbers indicate the presence of minor fixes or alterations which are still functionally compatible with versions of smaller fractional value. This simply means a document generated for one version will not trigger a PostScript error on a different version as long as the integer parts match.

❖ *Note:* Although there are few released versions of the Apple dictionary, in actuality the version numbers tend to change frequently, and the architecture is not guaranteed to remain constant. It is therefore recommended that your spooler not reply to or rely on specific versions of the dictionary.

Obtaining the font list

In the second or font list query, the printer driver asks for the list of all known fonts on the printer. The query begins with the string, "%?fontlist", followed by the necessary PostScript code and then the string "%?end". Figure 2-5 shows a font list query. The printer (and therefore your spooler) is expected to respond with a list of its fonts. The anticipated response differs according to the version of the driver. These differences are described below.

```
MSC NNNN  
ART: NN x NN  
16.5 picas text to FN b/b
```

Figure 2-5
Font List Query

Font list response for release 1 drivers: Release one LaserWriter drivers (with version numbers less than 3.0) expect the response to this query to be a sequence of strings each terminated by a carriage return. Each string specifies a single font (from PostScript's FontDirectory) which has been coordinated (see Chapter 3) and which is available on the printer. The string sequence is terminated with a string containing the single character '!', ASCII hex 2A. Only coordinated font names should be sent in response to this query.

```
% The query  
md begin  
%?fontList  
lst  
%?end  
end
```

Fig 2-5
Font List Query

pg. 2-38.1

◆ *Note:* Each font string transmitted by AppleTalk is encapsulated in a single packet (accomplished by executing a PostScript flush operator on the printer). This is the recommended convention for all strings the driver receives from the spooler, because it must sift meticulously through all responses to distinguish between status and data.

Font list response for release 11 drivers: Later releases of the driver (with version numbers 3.0 and greater) expect a similar sequence of font names in response to the font list query, as described above for earlier drivers, but with one difference: both coordinated and non-coordinated font names are valid responses. The driver will recognize the difference between the two types of font names by the vertical bar '|' character at the beginning of coordinated names, as seen in figure 2-6, which shows the response to the font list query in figure 2-5.

MSC NNNN
ART: NN x NN
16.5 picas text to FN b/b

Figure 2-6
Font List Query Response

Both the coordinated and non-coordinated name of a font can be defined in the dictionary. Your spooler should be able to distinguish between the two types of names, and it should never respond to a font list query with the non-coordinated name of a font that has already been coordinated. When the LaserWriter driver receives non-coordinated font names in response to the font list query, it attempts to coordinate them before proceeding with the document body.

% Sample List for LaserWriter Plus

NewCenturySchlbk-Bold

Times-BoldItalic

Palatino-Italic

Bookman-Light

Helvetica-Narrow-Oblique

Helvetica-Bold

Helvetica-Narrow

Courier-Oblique

Times-Bold

Times-Roman

NewCenturySchlbk-BoldItalic

Palatino-Bold

Palatino-Roman

_____ Courier

% A sample

pre-initialized font.

ZapfChancery-MediumItalic

ZapfDingbats

_____ Seattle

Courier-Bold

AvantGarde-Book

Palatino-BoldItalic

Helvetica

Bookman-LightItalic

Times-Italic

Courier-BoldOblique

NewCenturySchlbk-Italic

NewCenturySchlbk-Roman

Helvetica-Narrow-BoldOblique

Bookman-Demi

Helvetica-Narrow-Bold

Helvetica-BoldOblique

Helvetica-Oblique

AvantGarde-Demi

Bookman-Demitalic

Symbol

Fig 2-6
Font List Query Response

AvantGarde-BookOblique

AvantGarde-DemiOblique

*

Pg. 2-39.1

Coordinating non-coordinated fonts: One of the major new features of the 3.0 driver is the ability to configure fonts as necessary for any given document. If this requires downloading a font or coordinating a font, the driver will initiate this process. Two conditions occur to signal the driver to coordinate a font. First, the font must be present (along with its 'FOND' resource info) on the Macintosh that initiated the print job. Additionally, the name of that font in the font list query must flag it as non-coordinated.

When the driver determines (from the font list response) that it must coordinate a font, it generates the appropriate PostScript text to coordinate the given font, rename it, and make it semi-permanent on the printer. (The driver will also coordinate a font to be downloaded with a document, but in this case the font information will be part of the document and thus transparent to any intercepting spooler.) Figure 2-7 shows the PostScript code generated by the driver to coordinate the fonts listed in figure 2-6.

```
MSC NNNN  
ART: NN x NN  
16.5 picas text to FN b/b
```

Figure 2-7
PostScript Code for Coordinating Fonts

Coordinating fonts: The process of coordinating a font is very font-dependent and dynamically generated; trying to manage it through a spooler may become very cumbersome. There are two possible approaches that you can take to font management and replying to a font list query with your spooler.

```

% Make it stick
serverdict begin 0 exitserver
md begin
% General format is...
% /Coordinated-Name /Previous-Name MacEncoding-Vector-flag rf
/_____Times-Roman /Times-Roman T rf
/_____Times-Bold /Times-Bold T rf
/_____Times-Italic /Times-Italic T rf
/_____Times-BoldItalic /Times-BoldItalic T rf
/_____Helvetica /Helvetica T rf
/_____Helvetica-Bold /Helvetica-Bold T rf
/_____Helvetica-Oblique /Helvetica-Oblique T rf
/_____Helvetica-BoldOblique /Helvetica-BoldOblique T rf
/_____Courier-Bold /Courier-Bold T rf
/_____Courier-Oblique /Courier-Oblique T rf
/_____Courier-BoldOblique /Courier-BoldOblique T rf
/_____Symbol /Symbol F rf
/_____AvantGarde-DemiOblique /AvantGarde-DemiOblique T rf
/_____AvantGarde-BookOblique /AvantGarde-BookOblique T rf
      .
      .
      .
/_____Palatino-Roman /Palatino-Roman T rf
/_____ZapfChancery-MediumItalic /ZapfChancery-MediumItalic T rf
/_____ZapfDingbats /ZapfDingbats du fe
% Redefine encoding vector for above font.
% encoding-position, name, operand.
128 /a89 ce
129 /a90 ce
130 /a93 ce
131 /a94 ce
132 /a91 ce
133 /a92 ce
134 /a205 ce
135 /a85 ce
136 /a206 ce
137 /a86 ce
138 /a87 ce
139 /a88 ce
140 /a95 ce
141 /a96 ce
nf

```

Fig. 2-7.
 PostScript Code for
 Coordinating
 Fonts

One approach is to have the spooler reserve the font coordinating and downloading functions to itself, and return only coordinated font names in a font list. In this approach the driver never sees a non-coordinated font on the printer. Therefore, the driver never coordinates a font except when downloading a font required by a document but not returned in the font list. (The LaserWriter driver assumes that this approach is in effect when the Command-H option is used to send a print job to a text file.)

A second approach requires less intelligence on the part of the spooler and may be more independent of external changes. In this approach the spooler can impersonate the printer, capturing PostScript text from the driver to help it in its work. To impersonate the printer, the spooler can assume the name of the printer, and remove the name of that printer from the list of printers available on the network. To do so, the spooler can change the product name if it's an original LaserWriter or the AppleTalk name if it's a LaserWriter Plus. The spooler essentially becomes transparent to the font coordinating and downloading process, but passes relevant queries onto the printer and replies on its own whenever possible.

❖ *Note:* It is also possible for a spooler to appear as a device other than a Macintosh printer device. For such a device to be recognized or manipulated by the Chooser, it must conform to the Chooser device standard. In this form the spooler either can be made independent of available network printers, or can be used as a replacement for them.

In this second approach the spooler waits for the first font list query from a connecting workstation and captures it for later use. It then passes this query on to the client printer. When it receives a reply, it notes the font names and caches them away internally. Then it passes this information back to the inquiring workstation and waits for any further response from the workstation. When any font coordinating information follows from the workstation, the spooler captures this information and passes it onto the printer as a regular job. The spooler can then make font list queries of the printer periodically (perhaps after every printing job completes), capture the results, and then, when it receives workstation queries, simply return its own cached list of fonts, always reflecting the most recent status of the printer being observed.

The danger of this second approach is that the spooler's font list can get out of synchronization with that of the printer. (This synchronization problem can be especially troublesome for spoolers that do not capture the printer and isolate it from the network.) If the spooling administrator ensures that no permanent fonts can be added to the printer while the spooler is in operation, this problem can be minimized.

Providing your own PostScript dictionary

An application can now supply its own document header (PostScript dictionary) either permanently (till the printer is powered off) along with the Apple Laser Prep dictionary, or temporarily with each document to be printed.

The recommended method for downloading your own PostScript dictionary is to supply it on a once-per-document basis. This not only happens very quickly but it minimizes interference with the Apple PostScript dictionary, prevents unnecessary consumption of valuable virtual memory space on the printer, and allows more downloadable fonts to be used more abundantly by other applications. The method for accomplishing this is to provide a resource in your application of type PREC and ID 103, decimal, which consists of text data (including all necessary carriage returns or line feeds). When printing occurs the driver will look for a resource of this type and ID and, if it finds it, will send it to the printer during document open time. It is your responsibility to make this text syntactically correct.

Warning Under no circumstances should your application issue a PostScript `exitserver` operator.

The other method of providing your own PostScript dictionary is permanent (until power off). Your application must provide a 'PREC' resource, ID 201, which contains a set of parameters pointing to an unlimited number of resources that contain the actual PostScript code. This approach also allows for data to be in hex.

Resource 201 contains a pair of integer items for each separate resource to be downloaded. When the driver initializes the LaserWriter (and only then), it downloads its own PostScript dictionary. Then it looks for resource ID 201. The driver does not open any resource files, so this resource must be in an open resource file. The driver uses the first word of each pair in the resource as the ID of another resource of type 'POST'. The driver sends that resource to the printer, as text if the second integer of the pair is a 1 or as hex data if the second integer is a 0.

There are two special types of these pairs; if the first word of the pair is an integer 0 or 1 then there is no second word of the pair, and the first word is taken as an end-of-file indicator. An integer 1 in the first word of the pair indicates a temporary end-of-file: the file is closed, but the AppleTalk connection is not broken. An integer 0 in the first word indicates the absolute end of the dictionary data; this type of end-of-file indicator must be immediately preceded by a temporary end-of-file. It is the responsibility of the application to perform a PostScript `exitserver` if it is required in this mode.

Localizing the printing process

The Macintosh human interface is designed to allow applications to be localized for users across national and cultural boundaries. By viewing information as resources, the system allows you to modify much of interface accurately and easily. This is true of the printing process as well: all the relevant messages and menu items are available as strings in the driver that you can localize by modifying the appropriate resources. The only string not editable is the version string in the dialogs. The version string is in the code to preserve version identity. The strings that you will need to modify are shown in table 2-8.

Table 2-8
Localizable Resources

Resource	ID	Explanation
'PREC'	109	Pascal-style string pairs used in displaying status messages read from the printer.
'STR '	-4096	'STR 'strings are used by the Chooser to inform the user of available printer options. This string is the printer device name .
	-4095	Singular form of the printer device name.
	-4094	Plural form of the printer device name.

- 4091 Select string for the select-a-particular-printer-of-this-type option.
- 'DITL' -8191 Job dialog button and selection items. These may be modified as necessary.
- 'DITL' -8192 Style dialog items.
- 'DITL' -8181 Alert text.
- 'DITL' -8179 Printer mis-match alert. The printer designated in the print record is not of the proper type, e.g. an imagewriter print record has been detected.
- 'DITL' -8160 Zoom or reduction value selected by the user is out of range. The maximum or minimum is substituted automatically, and the user is alerted that this will happen.
- 'DITL' -8159 No Laser Prep file available on the Printer or for downloading. Printing cannot continue until the user inserts a disk with a valid Laser Prep file into the internal disk drive. If there is a disk already in the internal drive, it is ejected.
- 'DITL' -8152 This alert cannot be used.
- 'DITL' -8151 The Laser Prep file on the Printer and the Laser Prep file for this version of the LaserWriter driver are not the same. The printer must be powered down to be re-initialized.
- 'DITL' -8149 The help screen for the LaserWriter job dialog. These strings may be altered as necessary. Each string is in a separate rectangle for ease in formatting and appearance. You may wish to completely alter this Alert.
- 'DITL' -8150 No name has been chosen in the Chooser Desk Accessory. Consequently, the host cannot access a printer on the AppleTalk network.
- 'STR ' -8191 The name of the default spool file (normally *PostScript*) when Command and H keys are held down at the start of the print job.
- 'STR ' -8160 "Looking for LaserWriter <name>." The sequence "<>." should not be moved from the end of the string. This may cause translation difficulties in some languages, but the code expects the string to be of this form.
- 'STR ' -8159 "Creating PostScript File." The string in the status dialog box when Command H and Command K options are specified.

Changing printer messages

'PREC' string pairs are used to construct messages from the printer. For example, in the error message "PrinterError: *reason*" the error type would be matched to a pair via its first string. Consequently, this first string must not be altered in any way. The first string of the pair serves as a key to the second string. This second string is the one that would actually appear in the message in place of the word *reason*. It is this string you should translate.

You can add any number of string pairs to this list, do not remove any of the original pairs. The end of the list is indicated by a pair whose first string is empty. The strings sent over AppleTalk by the printer are as follows:

```
"job: ...; document: ...; status: ...; source: AppleTalk"
```

The values for the variable fields are as follows:

- possible values for `job` are infinite
- possible values for `document` are infinite
- possible values for `status` are busy, waiting, or idle. These are modified to read processing job, preparing data, and starting job, respectively.

The other forms of messages are:

- "Error: error message ; OffendingCommand: operator"
- "PrinterError: reason"
- "Flushing: rest of job (to end-of-file) will be ignored"
- "exitserver: permanent state may be changed"

Changing Laser Prep messages

Some messages are generated by the LaserPrep file which are of the form:

```
"|0|", "|1|", "|2|", ...
```

The correct interpretation for these strings is already in the string pair list and the second string of the pair may be translated as needed. You should never move the string pair beginning with "out of paper" from its position in the list.

Using PostScript code in a document

Resource 'STR' -8188 contains the string "PostScript Escape", representing the PostScript escape font. If this string is changed to the name of a font in the system resource file, when a document is sent to the printer, all text in that font will be sent to the printer as PostScript rather than as text to be printed. In this way the user has the ability to send special-effect PostScript code to the printer. The following restrictions apply to escape font text:

- All the text must be on one page; it cannot overlap onto an adjoining page.
- The text should not be on the same line as text in any other font.
- The QuickDraw characteristics of the text, such as pen positioning, style, and font size have no effect on any other printed attribute of the document.

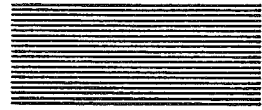
Intercepting PostScript files

The user can intercept the PostScript version of a document to be printed. This is done by holding down the Command and H keys immediately after clicking on the OK button in the job dialog box. The code is written to a file named PostScript which can be edited with a standard editor or word processor in text only mode.

The user can also intercept the Laser Prep dictionary code with the PostScript code for his document. This is done by holding the Command and H keys immediately after clicking on the OK button in the job dialog box. This code is also written to the file named PostScript.

This feature makes no provision for renaming such files. If a file named PostScript already exists, it will be overwritten.

Warning



Chapter 3



Working with fonts

Overview

Like traditional publishing systems, the Macintosh/LaserWriter desktop publishing system uses fonts to display text. Although fonts for both types of systems are used to produce the same effect, they are alike in almost no other way. Fonts for the Macintosh and the LaserWriter are software entities that describe how characters should be formed. Even fonts for the Macintosh can differ greatly from those for the LaserWriter. Fonts designed to be displayed on the Macintosh are known as screen fonts. Fonts designed to be printed on the LaserWriter are called printer fonts. The LaserWriter driver matches these two types of fonts together to achieve a publishing system whereby "what you see is what you get." This chapter describes the requirements for matching and using fonts in this system, including classifying, naming, and downloading fonts.

Understanding font terminology

As a result of the dynamic evolution of publication methods, terminology used to describe the printing process and printed text has become very confusing. The advent of desktop publishing has unfortunately added to this confusion.

❖ *Note:* The "Font Manager" chapters of *Inside Macintosh, volumes I and IV*, give a detailed introduction to fonts.

To avoid as much confusion as possible, table 4-1³ presents some common terms and their meaning, as used in this chapter.

Table 4-1³

Font Terminology

Font Family	A generic set of characters, including all styles and sizes of those characters. The Geneva font family, for example, would include 12-point Geneva italic characters and 36-point Geneva bold characters.
Font	A specific set of characters in a single size and style, for example, 12-point Geneva italic. A particular font may be <i>intrinsic</i> or <i>derived</i> .

Leading	The space between lines of text. The term—pronounced <i>leading</i> —is derived from the lead strips typesetters used to separate rows of type.
Size	The vertical measurement of a font in points, equal to the height of the font rectangle plus the leading.
Style	The characteristics other than size which uniquely define the fonts of a single font family. Geneva bold and Geneva italic are two styles of the Geneva font family.
Intrinsic Font	A font whose characteristics are entirely defined in a 'FONT' resource. The plain-style font of any family is an intrinsic font: other styles may or may not be intrinsic. An intrinsic font can be used by QuickDraw or the LaserWriter without modification.
Derived Font	A font whose characteristics are partially determined by modifying an intrinsic font. A derived font might be one whose characters are scaled from an intrinsic font to achieve a desired size, or slanted to achieve an italic style.

About screen fonts

Any black and white image can be displayed as a collection of black and white dots. These dots are called picture elements, or pixels. Fonts for electronic publishing are really data structures that specify arrangements of pixels. The Macintosh displays pixels on its screen as tiny squares, at 72 per linear inch. (This size coincides with the definition of a printer's point—1/72nd of an inch.) In contrast, the LaserWriter can print 300 pixels per inch.

Macintosh screen fonts are specified as bit maps: a bit image of each character in the font is stored in memory. When the user types that character, each pixel of the character is drawn on the screen as specified by the corresponding bit in memory. The bitmap approach works well for a screen display, but it does have drawbacks. Each character for each font must be represented by its associated bit map, which can require large amounts of memory as you add fonts to the system. Enabling the user to freely specify different sizes of characters requires either more bit maps or a mechanism for enlarging and reducing bit images. In reality, both methods are used by the Macintosh for screen displays.

The Macintosh provides a wide variety of font families with a well defined set of styles uniformly available across all families. For

each Macintosh font (screen font) the user may select up to seven distinct styles provided by QuickDraw:

- plain (often called Roman)
- bold
- italic (or oblique)
- underline
- outline
- shadow
- small caps

Usually the plain style is the only style available in an intrinsic font, and the other six are derived. (Some applications provide even more styles than QuickDraw does.) In some cases, and especially with the new Font Manager for the 128K ROM, screen font styles other than the plain style are intrinsic rather than derived.

About printer fonts

Printer fonts are defined in a very different manner from screen fonts. The image of a character is defined as a series of Bezier curves, or B-splines. These curves are stored as mathematical constructs that form the outline of the character, which is then simply filled in. There are several advantages of this type of character definition:

- Drawing the image of the character takes much less time than constructing the image from a bit map.
- The sizes of the curves are easily reduced or enlarged, producing a clear image of the character, regardless of its size.
- Since one definition specifies all sizes of a character, less memory is required to store many sizes of a font.
- The definition is device-independent. It can be reproduced on any PostScript printer. The resolution of the printer determines the quality of the printed image.

The LaserWriter contains a number of built-in printer font families in its ROM, including several intrinsic bold and italic fonts. In addition, the LaserWriter has the capability of accepting new fonts that may be downloaded to it either on a once per document basis (temporary downloading), or for use on any document until the printer is powered down (permanent downloading). The number of downloadable fonts that can coexist on the printer at any one

time is limited only by the available memory, but, depending on the size of the fonts involved, is usually between 5 and 10.

At the beginning of every document the LaserWriter printer driver queries the LaserWriter, asking it to list all the fonts it has. The driver stores information about all these fonts in a temporary font cache. The driver parses this cache whenever a new font is encountered in a document. If the desired font is in the cache, the printer is switched to that font. If the desired font is not in the cache, the driver initiates a disk search for a font file to download to the printer. If an intrinsic font is found, it is downloaded to the printer. If one is not found, the derived (bit map) version is created and downloaded. In either case, the downloaded font name is also entered into the cache.

When the driver requests the font inventory from the printer, it receives information about permanently downloaded fonts as if they were built-in fonts. It can then use permanently downloaded fonts in the normal manner, provided the downloaded font is consistent with other printer fonts. (To be consistent, the downloaded font must be defined and classified correctly and must be matched through an appropriate 'FOND' resource to a corresponding screen font on the Macintosh.

Downloadable fonts are not limited to splines or outline forms but can be arbitrarily defined by your application or the user as long as they conform to Macintosh and PostScript conventions. Bitmap fonts are just one example of fonts that can be defined and downloaded. As mentioned above, the LaserWriter driver can temporarily download a screen bitmap font automatically when that font is encountered in a document but does not exist on the printer.

It is at this juncture that the differences between screen and printer fonts becomes painfully obvious. If the user selects a character in a size that is not defined in the screen font's bit map, QuickDraw attempts to resize the character by a method called scaling. The result of this process can sometimes be less than spectacular; the character can be significantly distorted, with rough curves and jagged edges. It is this image that is sent to the LaserWriter. The LaserWriter then accurately reproduces the bit map on the printed page.

Matching screen and printer fonts

The crux of implementing a desktop publishing system is the process of properly matching screen fonts with printer fonts. In the

typeset world a font family may contain anywhere from 1 to perhaps 50 different styles. The Macintosh can provide upwards of 64 styles, but not necessarily the same styles as in typeset font families. For example, the Macintosh has only one level of bold weight per family, whereas some typeset families may have more than four weights (light, demi, bold, heavy, ultra, etcetera).

Font matching is further complicated by the large number and many types of characters that can exist in a font. For example, Japanese kanji fonts typically contain more than 3,000 intricate characters, and Arabic fonts include ligatures and characters whose forms change depending on their context.

Additionally, some styles may be derived in some fonts yet intrinsic in others. To make matters worse, a derived style in one font family may be implemented differently from the same derived style in a different font family (this is true particularly for outline and shadow styles).

In order for the printer driver to be able to handle all these variations correctly, each font must be marked by its own characteristics. These characteristics determine the font class; when they are properly established, the font is considered *classified*. A font that is not classified is unusable.

Classifying fonts

Font classes are specified differently for the different versions of the driver. In the first version released, the font class is specified in the 'FONT' resource for the font. In the second release, the font class is specified in a *style mapping table*. For both releases; the font class comprises two basic types of information about the font: the *style implementation method* and the *character set encoding* scheme.

Style implementation

A style implementation is a set of methods used to obtain font styles. It may include both inherent and derived style types. Each font in any given class implements its styles according to these methods defined by the class. The class also helps the driver distinguish between inherent and derived styles.

Character set encoding

Character set encoding arises from the implicit difference between the Macintosh character set and the Adobe StandardEncoding vector. PostScript allows characters within encoding sets to be rearranged at your discretion to match other sets (such as the Macintosh character set), but it does not allow characters to be swapped between different character sets. As an example, the Adobe Standard Encoding for the Times Roman and Helvetica fonts does not normally contain the characters '®', '©', '™', and '☺', but they are available in the encoding for the Symbol font. Since these characters are present in the Macintosh character set, the printer driver must switch automatically to the Symbol font to print these characters whenever it encounters them. Fonts that require this type of character borrowing are called re-encoded fonts and fall in a different class from those that don't.

❖ *Note:* Although re-encoded printer fonts do not contain these characters, the associated screen font must. (Also note that when a font switch of this type occurs, the design appearance of the switched font is not altered to match the initial font family; for example, sans serif does not carry over into the serif Symbol characters.)

It is also possible for a font to incorporate other encodings which are neither the Adobe nor the Apple character set. A font with its own character coding defined intrinsically presents no problem. In such a case the screen font is made to match the printer font in each character position, and the font creator can assign character positions within the font arbitrarily.

There are some printer fonts (such as Zapf Dingbats), however, which contain additional characters that are not assigned to any particular character position. (PostScript, allows a font to contain an arbitrary number of characters—even more characters than the number of available encoding positions.) These characters can be assigned to normally unassigned positions in the encoding vector or to character positions already defined. In order for such fonts to be downloaded by the driver, the encoding information must be available to the driver whenever it is ready to make the new character encoding assignments.

The font characteristics described above determine the class of a font. The format of the class entry to the style mapping table varies according to the version of the printer driver number.

Classifying fonts—release I

In its first release, the LaserWriter driver had to process only a limited number of fonts; therefore, font classifications were neither very extensive nor comprehensive. Nevertheless, fonts may be added in the original release software, provided that the fonts fit into the limited classification set space available and are downloaded using the permanent downloading technique.

Table 4-2
Font Classification—Release I

- 0 The default class. Screen fonts without corresponding printer fonts fall into this category. These fonts are not re-encoded.
- 1 The standard class of Adobe fonts. Class 1 fonts are outlineable and re-encoded. Note that Courier, while a standard Adobe font, does not fit in this class because it is not outlineable in the PostScript sense.
- 2 Class 2 fonts are outlineable but not re-encoded. There are no bold or italic faces for this class. The Symbol font is a good example of a font in this class.
- 3 Class 3 fonts are re-encoded but not outlineable. This class is identical to class 1 except for the outlineable property. Courier fits into this class.
- 4 Class 4 fonts are both re-encoded and outlineable, as are class 1 fonts. But there are no other bold or italic faces available for the family. Italic is simulated by obliquing, and bold by increasing the point size slightly.

In the first release, the font class is determined by the Laser Prep header at the time it is loaded into the printer. Your application can add new fonts in one of two ways. The first is to download the font to the printer prior to the header, in which the font must be non-coordinated (see Coordinating fonts, below) and must belong to class 1 above. Alternatively, the font can be downloaded to the printer after the Laser Prep header, in which case it can belong to any of the above classes, and must be pre-coordinated. It is possible to add classes, but it is a difficult process and it is not recommended.

Classified screen fonts are matched with printer fonts by a process called *style mapping*.

Style mapping

The new LaserWriter driver (versions 3.0 and later) uses a process called style mapping. It uses a style mapping table (part of the 'FOND' resource) to match screen and printer fonts. This table contains the font class identification, character encoding information, and a mechanism for obtaining the name of the appropriate printer font. That font can be a built-in printer font or a downloaded (either permanently or temporarily) font. Figure 4-1 shows the structure of the style mapping table.

*** figure 4-1 Style mapping table ***

Classifying fonts—release II and later

The second and later releases of the printer driver provide a far greater number and variety of font classes. With these later releases, however, the classification is not provided either by the printer or the print driver. Font classification is a property of the font itself and must be available to the printer driver through the 'FOND' resource either at the time the font is downloaded or at the time the font is accessed.

❖ *Note:* 'FOND' and 'FONT' resources are maintained and controlled by the Macintosh Font Manager. Your application should never perform a release resource, detach handle, or purge handle operation on any 'FOND' or 'FONT' resource. Applications may, however, call `FMSwapFont` or `SetFontLock` along with other Font Manager calls to avoid potential Font Manager problems.

In these later releases, the class is available as the first entry in the style mapping table in the form of a sixteen-bit integer. Table 4-3 shows the bit assignments of this integer.

Table 4-3
Font Classification (Release II and Later)

Bit	Meaning
0	Set if font name needs coordinating.
1	Set if Macintosh vector re-encoding scheme is required.
2	Set if font is outlineable by changing PaintType to 2.
3	Set to disallow outlining simulation by smear and whiteout.
4	Set to disallow emboldening by smear technique.
5	Set if emboldening is simulated by point size increase.
6	Set to disallow obliquing for italic.

Style Mapping Table

2	Font Class
4	Offset to Encoding Table
4	Reserved
1	Suffix index for Style Code 00
1	Suffix index for Style Code 01
1	Suffix index for Style Code 02

1	Suffix index for Style Code 47

pg. 3-9.1

- 7 Set to disallow automatic simulation of condensed style.
 - 8 Set to disallow automatic simulation of expanded style.
 - 9 Set if re-encoding other than Macintosh vector is required.
 - 10-15 Reserved
-

In the absence of any defined class, the class definition defaults to class 0, which has default settings that indicate that the bold, italic, condensed, and expanded styles should be derived from the plain font. Intrinsic fonts, either downloadable or printer-resident, are assigned classes which prevent the derivations from occurring. Thus the distinction between derived and inherent styles is apparent. Bits 10 through 15 of the class definition integer are reserved; your application should not use them.

Encoding character sets

If the font class indicates that the font must be re-encoded, the second and third words of the style mapping table are used as an offset to a *character set encoding table*. Figure 3-2 shows the format of this table.

***Figure 3-2. Character Set Encoding Table ***

If the encoding for the font differs from the conventional Macintosh character set, does not contain its own font encoding, or otherwise needs to be altered, the new encoding vector is derived from this table. If the offset in the style mapping table is empty, there are no extra encoding requirements.

The encoding table itself contains an integer length field followed by an entry for every character position requiring re-assignment. Each character position entry consists of a single byte character code and a Pascal string comprising the PostScript character name key (excluding the normal preceding slash). Note that fonts which require this re-encoding scheme must also be flagged as requiring coordination as well as re-encoding.

❖ *Note:* The offset to the character set encoding table is a relative offset from the beginning of the style mapping table, not from the beginning of the 'FOND' family definition record.

Specifying printer font names

Because there are so many more font weights available on the printer than on the Macintosh, all printer font families are first

Character Set Encoding Table

2	Number of entries (1 word)	
1+String Size	Char Code	Character Name String
1+String Size	Char Code	Character Name String
1+String Size	Char Code	Character Name String
1+String Size
1+String Size	Char Code	Character Name String

Example:

Char Code Character Name

\$90 a89

\$A8 diamond

fig 3-5

pg. 3-10.1

~~Page 3~~

subdivided into subsets. There may be one or more subsets per family, including the subset that encompasses the entire family. Each subset is given a unique font name (usually a derivative of the font family name).

The user selects a screen font from the Macintosh font and style menus. For each selectable screen font, each printer font family subset contains at most a single font into which that Macintosh font will be mapped. Subsets can vary in size and can overlap within a family, but they can never overlap another font family nor exceed more than one font per Macintosh font.

For every font that is available in a given style there is a non-zero entry in the corresponding position in the table. That entry is an index that points to the *style name table*, from which a unique printer font name is derived.

On the LaserWriter, many derived styles can be generated automatically by modifying existing fonts, as determined by the font class. Underline, shadow, and outline styles are derived this way and are relatively easy to generate dynamically. On the other hand, bold, italic, and bold italic styles are often not so easy to generate. Thus these styles are usually obtained from intrinsic fonts. Some font families have missing fonts: fonts for which there is no supplied face and for which no derivation is required. For missing fonts, the corresponding style name index entry in the style mapping table is zero. If such a font is selected, the plain face is used instead, without modification.

The style name table is a string list containing the information necessary to produce the font name for that style. Entry number one in the string list contains the base font name to which zero, one, or more suffixes, also contained in the table, may be appended to derive the full font name. Following the base font name are index strings which provide access to font name suffix strings. The string list thus contains both index lists (in the form of strings) and suffix strings. Figure 3-3 shows the format of the style name table.

Figure 3-3. The Style Name Table

Style Mapping Process

Style mapping occurs by initially matching the selection from the Macintosh font menu to a 'FOND' resource which contains the mapping table for the given printer font family subset. The selected style is matched with a style code.

Style Name Table

String Count	
Full Base Font name excluding suffixes	
Suffix index list for suffix index 2	}
Suffix index list for suffix index 3	
Suffix index list for suffix index 4	
etc.	
Full suffix #1	←
Full suffix #2	←
Full suffix #3	←
etc.	

Fig 3-3

The normal Macintosh style bits are as shown in table ^{2 3} 4-4.

^{2 3}
Table 4-4
Macintosh Style Bits

Bit	Meaning
0	bold
1	italic
2	underline
3	outline
4	shadow
5	condense
6	extend

Style Mapping, however, omits the bit assignment for underline. to provide the style code format as shown in table - 3-5.

Table 3-5. Style Code Format

The bit list in Table 3-5¹ shows the numeric order of style entries in the style mapping table; it does not represent any actual data quantity. Because it resembles the actual Macintosh style assignment in table 4-4, it is important to notice the differences. It has been modified solely to optimize storage space.

For each style code there is an entry in the mapping table to provide the offset in the style name table of an index number. The index number points to a numbered string in the table. This first string is itself an ordered list of index numbers which point to actual suffix strings in the same string list. The actual suffix strings are appended in the given order to the base font name to produce the full font name.

When a style must be derived, as specified by the font class, it is derived from the font specified in the style mapping table for that style. Thus the entry for the outlined italic style may contain the font name for the italic style, and the outlining will be derived (if so specified in the class) from the italic face. As described above, when there is no entry (zero entry) for a given style in the style mapping table, that style will be derived from (or replaced by) the plain style entry in the style mapping table. (The plain style entry, index 0 or style 0, must, therefore, never be empty or zero; that is, the 0 style index position must always point to some name string). It is permissible for different style entries to point to the same font name.

Naming fonts

The name that a user sees when he selects a font from the font menu in an application is the name of either the 'FONT' or 'FOND' resource. This name is used to match the font name on the printer. When a font is classified and named properly, it is considered to be *coordinated*. A font that is not properly classified or is not properly named (even though it may be syntactically correct to PostScript) is said to be non-coordinated. Whether a font should be coordinated or not depends on how it is to be downloaded to the LaserWriter.

Release I font names

In release I of the printer driver, a font designed to be downloaded to the printer prior to the Laser Prep dictionary cannot be downloaded after the dictionary, and vice versa. The names for different types of fonts are also derived differently.

If a font is to be downloaded before the Laser Prep dictionary, the font name must not exceed 30 characters in length and must not include any style suffixes other than "Bold", "Italic", "Oblique" or "Roman". All such fonts will fall into class 1.

❖ *Note:* These fonts must also be non-coordinated

Fonts that are downloaded after the Apple dictionary must first be coordinated. A coordinated printer font name begins with a 6-character prefix followed by the name of the font that appears in the Macintosh font menu, with spaces and minuses removed. The prefix format is |—#, where the vertical bar marks the font as being coordinated and the # is the class number of the font. Note that the class number is an ASCII character, not a binary code. The four hyphens are place holders for various intrinsic style attributes of the font. The first hyphen is replaced by the letter *B* if the font is bold. The second hyphen is replaced by *I* if the font is italic. The third hyphen becomes *O* for outline and the fourth becomes *S* for shadow. Outline and shadow intrinsic fonts are created automatically in release I.

Release II and later font names

Coordinated font names in later releases are prefixed by the 7-character string |_____. Note that although the vertical bar

has been retained, the hyphen characters have been changed to underscore characters (hex 5A), and the class number has been eliminated from the font name altogether. The underscore character positions, though still valid for *B*, *I*, *O*, and *S*, are reserved for Apple use. These characters are never changed by the application.

The name following the prefix is the name of the font as it appears in the style mapping table. The name is the normal name for the font on the printer, including all suffixes. Hyphens are considered to be separate suffixes, and spaces are not allowed. The name does not necessarily match the name given in the Macintosh font menu, nor does it necessarily match other font names in the same font family. The name in the font menu is used to access the appropriate 'FOND' resource, which contains the style mapping table, which in turn points to the actual font name.

Downloading fonts

If the LaserWriter driver discovers a font in a document that is not available on the LaserWriter, it attempts to download the correct font to the printer.

Downloading with the release I driver

Release I of the printer driver does not support temporary (automatic) downloading. An application using release I of the driver is therefore responsible for downloading the font to the printer in a form that is useable by the driver as described in the Font Names section, above. A common method for doing this is to use PSDump to download the font permanently. The data format for such files used by PSDump is straight PostScript text. The PostScript conforms to the font definition as described in the *Adobe Font Manual* and *PostScript Language Reference Manual*. To ensure that the font is loaded permanently, the text must be preceded by the PostScript string "*password* serverdict begin exitserver", where *password* is the password number of the intended LaserWriter, usually 0.

Release II and Later Downloadable Fonts

Downloadable fonts for release II and later can be loaded on either a temporary basis (automatically), or permanently. The fonts need not be pre-coordinated, but if they are not, they will be coordinated from the information contained in the 'FOND' resource supplied with the screen font. Notice that every font which is intended for downloading must have a corresponding 'FONT' and 'FOND' resource on the Macintosh.

Downloadable Fonts for All Releases

For a font to be useable both with release I and with release II or later drivers, the font must be initially non-coordinated and must have a font name which does not contain any suffixes other than Bold, Italic, or Oblique. Its class must be class 1 for release I and class 95 (hex \$005F) for release II. The font must be loaded prior to the Laser Prep dictionary for use with release I; for release II and later, it can be loaded permanently before or after the dictionary, or it can be temporarily downloaded as required.

Temporary Downloading of Fonts

Temporary (or automatic) downloading of a printer font is available with release II and later versions of the printer driver. When a new font is encountered in a document, the driver cache is scanned for that font. If it is found, a switch is made to that font on the printer. If it is not found, a search is made in the 'FOND' resource for that font. If there is no 'FOND' resource, the bitmap version of the font is created. If the 'FOND' resource is found and it contains an entry in its style mapping table for the appropriate style, the root directories of all on-line volumes are searched for a file name which corresponds to that entry. If one is found it is downloaded; otherwise, the bitmap version is created.

The downloadable font file is a resource file whose name is derived from the first five characters (excluding the first four characters, "TTC-", if present) of the full base font name in the style mapping table of the 'FOND' resource. The first five letters are followed by the first three letters of every suffix required for that style of the font. The base name and suffix positions are determined by the positions of the capital letters appearing in the font name.

a. This method is required because the file name cannot exceed more than 31 characters due to restrictions in TFS.

◆ *Warning:* The font name must never cause generation of a file name which is longer than 31 characters.

The format of a temporary downloadable font resource file is described under *Downloadable File Format*, below

Permanent Downloading of Fonts

As discussed above, fonts may be downloaded permanently (i.e., until the LaserWriter is powered down) either before or after the Laser Prep dictionary is loaded. Both methods are described below.

Fonts Loaded Before Dictionary

Fonts can be downloaded on a permanent basis by using PSDump, if the font definition is preceded by the string "password serverdict begin exitserver". But for the font to be used by the LaserWriter driver there must be a 'FONT' and 'FOND' Macintosh resource which points to that font on the printer. Then when the font is encountered in a document, the entry in the driver cache will be noted and the 'FOND' resource will simply provide the class for the font. A printer font and Macintosh screen font pair without a corresponding 'FOND' resource may still be used by the print driver, but the Macintosh menu name must agree with the printer name (with spaces in place of hyphens), and the downloaded font must be pre-coordinated. Any style for such a font will be derived as though it were a bitmap font.

If PSDump is used to permanently download fonts, the fonts must be in PostScript text form (as opposed to the resource file form of temporary downloadable fonts). It is possible for users or the application to download the resource form of the font, but some utility that can accept the resource form must be used in place of PSDump.

Fonts Loaded After Dictionary

Unlike in release I, downloading a font after the Laser Prep dictionary in subsequent releases is performed exactly as downloading a font before the dictionary. No changes to the font are required.

Downloadable File Format

A permanent downloadable file must be in Postscript text if PSDump is to be used for downloading. The name and font definition must conform to the specifications above. This does not preclude the development of other methods for downloading, but if they are to remain compatible with the LaserWriter driver, the naming conventions and 'FOND' structure must be followed.

A temporary downloadable font file is an unbundled resource of type 'LWFN' whose creator is LWRT. The file is made up of several resources of type 'POST'. The ID's for these resources begin with the number 501 (decimal) and increment by one for each resource in the file. The number of resources is not limited. Each resource begins with a two-byte data field which contains the data type in the first byte and binary zero in the second. The rest of the resource is the data to be downloaded.

The possible values for the first byte of the two byte field are 0, 1, 2, 3, 4, and 5; their meanings are shown in table 4-6.

g³
Table 4-6

Temporary Font Resource Format

-
- 0 Ignore the rest of the resource (thus, a comment).
 - 1 The data is ASCII text.
 - 2 The data is binary and is first converted to ASCII hex characters before being sent.
 - 3 An AppleTalk end-of-file will be sent (maintaining the AppleTalk connection.) The rest of the resource data for this value is interpreted as ASCII text and will be sent following the AppleTalk EOF. If there is no additional ASCII text to be sent, the rest of the resource can be empty.
 - 4 The data fork of the current resource file is to be opened and sent as ASCII text. (This should be done only once.)

- 5 The end of the resource file; the data in the resource itself may be empty or ASCII text.
-

The second byte of all the two byte fields is reserved and should always be set to zero.

Since resource data is always loaded into memory in its entirety, it is recommended that the resources be kept small, certainly less than 2k. It is not necessary for the data in a resource to begin and end on any particular boundary, but text and binary data may not be mixed in the same resource. For RMaker users it may be helpful to set type 'POST' equal to type 'GNRL' when constructing these resources.

When the font data is downloaded, no interpretation of the data is performed and no prefix or suffix data is included in the downloading (except that non-coordinated fonts will be coordinated following the downloading, if so specified in the font class). The data in the file is presumed complete and self contained. The resource should never contain the "password serverdict begin exitserver" string. Each resource is downloaded in sequence beginning with resource ID 501 and proceeds in numerical order until there are no more resources available in the file or until a data type 5 is encountered.

If a font must be coordinated by the driver after downloading (as specified in the class) the driver will redefine the font on the printer using the name given in the 'FOND' resource for that style, re-encode it if required, and give it a new name prefixed with the string '|_____|' as described under *Font Names*, above. Thus the name in the 'FOND' and the normal non-coordinated name for the font must agree. The downloading process must define the non-coordinated font on the printer with the same name that the 'FOND' specifies. It will subsequently be assigned the coordinated name by the driver. (Adobe built-in and downloadable fonts are examples of non-coordinated fonts that must be coordinated before the driver can access them.)

Downloadable fonts have a great potential for creating disaster. They are essentially in control of the of the entire printed document. The LaserWriter driver tries to shield itself from misuse, but cannot cover every possible problem. For example, the driver cannot determine whether or not an error has occurred as a result of the download and simply assumes all is well. However, if an error does occur, it will be reported in the normal status window on the Macintosh screen. Any additional side effects that the font definition causes will be preserved over the rest of the document, so such fonts must be used with extreme caution.

Unfortunately, dynamic unloading of fonts (removing a specific font at will) is not easily achieved in PostScript except on a LIFO basis—the last loaded is the first unloaded. Because of this limitation dynamic unloading is not currently supported at all by the LaserWriter driver. Fonts downloaded permanently can be unloaded only by powering off the printer.

Downloading fonts from non-Macintosh hosts

Applications running on non-Macintosh hosts can download fonts to the LaserWriter by issuing the correct PostScript code. The method for doing so is outlined below. In this environment responsibility for managing fonts on the LaserWriter lies solely with the application.

Downloading fonts permanently

You can download fonts permanently by sending the PostScript font file as a separate job. You should send the following line of PostScript code immediately prior to the font file:

```
serverdict begin 0 exitserver
```

The 0 in this line is the default password for the printer. Your application should be prepared to handle printers with other passwords.

You should send the LaserWriter a control-D character sequence immediately after the font file, as an end-of-job indicator. The font will then remain in the printer until power-down.

Downloading fonts temporarily

You can download a font temporarily by simply sending it to the printer with the document file. The font code must appear in the document file before the font is required.

Accessing downloaded fonts

Once a font has been downloaded, it can be accessed through PostScript just as with any other LaserWriter font. Your application should be prepared to recognize a variety of fonts and to ensure that they are available to the user as required.

Checking for downloaded fonts

If your application does not maintain a current font list, it should check before sending each font to see if the font is already available in the printer. The following PostScript code will initiate a "yes" response from the LaserWriter if the font is in the printer, or a "no" response if it is not. (The word *fontname* should be replaced with the name of the font you are checking for.)

```
/scratch 100 string def           %create a scratch string
FontDirectory /fontname           %search FontDirectory for
                                   fontname
  ((yes))                          %if it's there put (yes) on the
                                   stack
  ((no                               %else put (no) on the stack and
  systemdict /filenameforall known %check for a file system at
                                   printer
  ((fonts/fontname) (pop pop (yes)) %if a file system is
  filenameforall) if               %present, search
  ) ifelse                          %for fontname; if found, replace
print flush                        %top of stack with (yes)
                                   %pop stack to buffer and flush to
                                   host
```

You can get a list of all fonts in the printer with the following PostScript code:

```
/scratch 100 string def           %create a scratch string
FontDirectory (pop == ) forall flush %send names in FontDirectory to
                                   host
systemdict /filenameforall known   %check for file system at printer
  ((fonts/*) (dup length 5 sub 5 exch
```

```
getinterval =) scratch filenameforall %if file system present, send
flush) if %names of fonts present to host
```

This will cause the list of fonts to be transmitted back to the host as ASCII text, one font name per line.

Determining available room for fonts

A limited number of fonts can be resident in the LaserWriter at a time. Before downloading a font, your application should check to be sure that there is room in RAM for it. The following PostScript code will return the number of bytes available in the LaserWriter RAM:

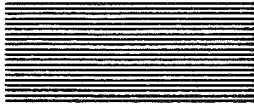
```
vmstatus exch sub = flush pop
```

If the number returned is greater than the number of bytes in the font file, there should be enough room to download the font.

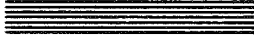
Making room for more fonts

Currently there is only one way to recover memory in the printer. This method involves using the PostScript command `save` to store the state of the virtual memory before a font is downloaded, and the command `restore` to return the virtual memory to its previous state after the font has been used. You may want to do this with each font, especially if the application allows a large number of fonts to be in use with a single document.

Warning Use the PostScript commands `save` and `restore` with caution: `restore` can erase some definitions.



Chapter 4



**Working in the printing
environment**

The LaserWriter is connected to the outside world by two serial ports: an AppleTalk RS422 connector and an RS232 connector. The communication channel established through these ports (over which the LaserWriter communicates with the host) is bidirectional. The LaserWriter can send as well as receive data. Communication over this channel is logically asynchronous. In other words, the LaserWriter is already to receive data, even while it is sending; it expects the host to do the same.

How the LaserWriter processes the information it receives over the channel is determined by its current operating mode:

- Batch mode is the LaserWriter's normal operating mode. In batch mode, a job consists of executing a single file containing a PostScript program. When the end-of-file indicator is received, the job is finished. The only data that the LaserWriter transmits to the host is that generated specifically by the PostScript print operator or by an error. If you are writing an application that prints on the LaserWriter, it will be working in this mode.
- In interactive mode the user can communicate directly to the LaserWriter from a Macintosh running MacTerminal™. The LaserWriter assumes the role of the powerful computer that it really is, and while still serving as a printer. This mode is useful for experimenting with PostScript and for using the LaserWriter as a general-purpose computer. If you need to test parts of your application, this mode can be handy.
- In emulation mode the LaserWriter is programmed to emulate other printers. The LaserWriter has a built-in emulator for the Diablo 630 daisywheel printer, which is widely supported by personal computer applications. If your application expects to print on a printer compatible with the Diablo 630 API interface, this mode will enable the LaserWriter to accept the control codes the application generates.

The LaserWriter has a four-position mode switch on the back. Together with some PostScript persistent parameters (see the PostScript Language Reference Manual), this switch controls the mode of operation and the communication protocol. The switch positions are listed in table 4-1.

Table 4-1
Operational modes

Switch setting	Mode effected
AppleTalk	PostScript batch mode operation: AppleTalk communication.
Special	Diablo 630 emulation mode: serial communication using the previously set parameters. (The default parameters are 9600 baud, parity ignored.)
9600	PostScript batch mode operation: serial (RS-232/-422) communication using the current communication parameters. (The default parameters are 9600 baud, parity ignored.) Note: Since these parameters can be reset under software control, the "9600" switch position may select a baud rate other than 9600.
1200	PostScript batch mode operation: serial communication via either of the two connectors, at 1200 baud, with parity ignored.

Changing the switch setting has an immediate effect: if a print job is in progress, it is terminated.

Using AppleTalk

AppleTalk is the network architecture that governs communication between LaserWriters and Macintoshes. The AppleTalk *Printer Access Protocol* (PAP) is a session-level protocol that allows a workstation on the AppleTalk network (typically a Macintosh) to communicate (through its PAP client) with the LaserWriter (through its PAP client). PAP is a connection-oriented protocol: in addition to managing the transfer of data, it performs the initiation, maintenance, and termination of logical connections. PAP uses other lower-level Appletalk protocols, notably the AppleTalk Transaction Protocol (ATP) and the Name Binding Protocol (NBP) to properly perform its duties. The Macintosh Printing Manager uses PAP in a way that's transparent to the application. You will only need to call PAP directly in order to perform supervisory or non-standard functions on the network.

❖ *Note:* The term *workstation* is used rather than *host* to refer to the Macintosh in the context of the network, to avoid a host of misleading statements.

To use the information in this chapter you should be familiar with the AppleTalk protocols, as described in the following documents:

- *Inside AppleTalk* (for introductory and background information).
- The "AppleTalk Manager" chapter of *Inside Macintosh* (for the Macintosh interface to AppleTalk).
- The "AppleTalk Developer's Notes for the //e" (for the Apple // interface to AppleTalk).
- ❖ *Note:* An Apple // equipped with an AppleTalk card can also use AppleTalk.

Connecting a LaserWriter to AppleTalk

Connecting a LaserWriter to an AppleTalk network requires an AppleTalk connector box with a DB-9 connector. A connector box with a 25-pin plug won't work, even though the LaserWriter does have a 25-pin socket.

Caution Before you connect a LaserWriter to an AppleTalk network, first turn off the LaserWriter and set the four-position mode switch to "AppleTalk". Never operate a LaserWriter connected to AppleTalk with the mode switch set to any other position—doing so may leave the LaserWriter in an inoperable state or even bring down the entire network.

A print job via AppleTalk consists of four phases—initializing the LaserWriter's PAP client, opening a connection between a workstation and the LaserWriter, transferring data, and closing the connection. Figure 4-1 shows the typical job cycle.

4

Initializing the printer node

When the LaserWriter first starts up and completes its internal initialization, it issues an `SLInit` call to its PAP, indicating that the printer is ready to accept print jobs from workstations. Just after the `SLInit` call, the LaserWriter's PAP client makes a `GetNextJob` call to prepare the LaserWriter to accept connection requests over the AppleTalk network.

Opening an AppleTalk Connection

A connection is a logical relationship between a PAP code entity (called a client) in the workstation and another in the LaserWriter.

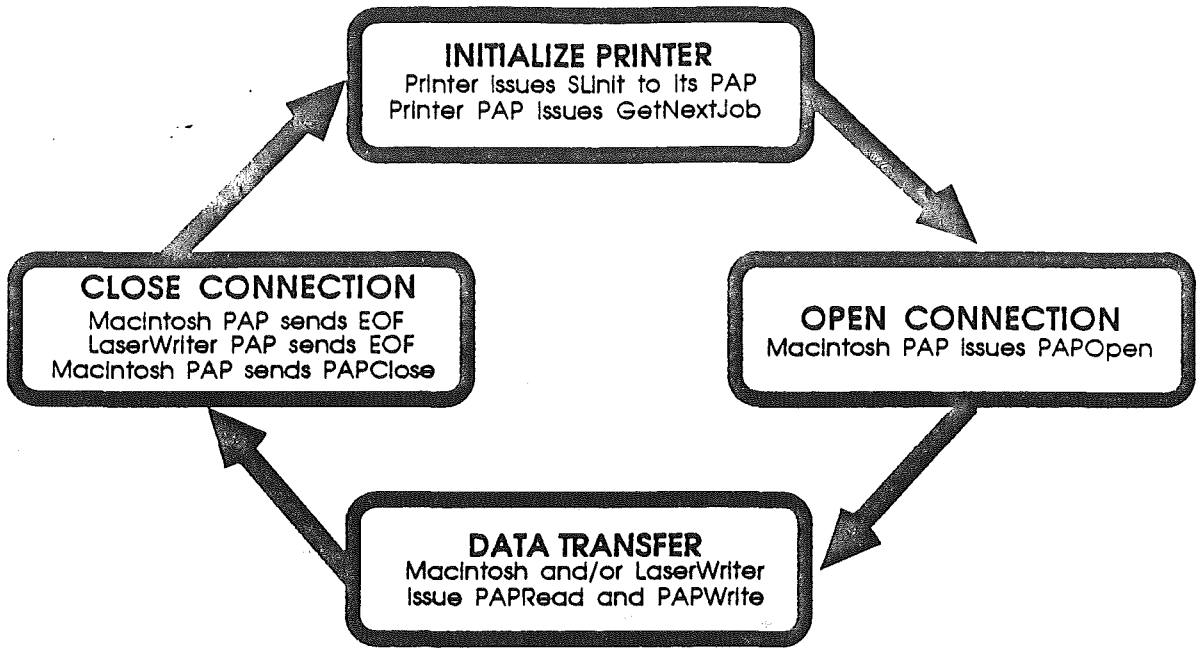


Fig 4-1. AppleTalk Print Job Cycle

To open a connection, a PAP client in a workstation issues a PAPOpen call (see the PAPOpen section, below), which initiates a connection-establishment dialogue with the LaserWriter.

When the LaserWriter receives and accepts the request, it executes a job using that connection as its standard input (only one job is executed per connection). At any given time, the LaserWriter has at most one open connection—while it's busy with that connection, it refuses any further connection requests. The LaserWriter PAP queues connection requests from the hosts in the order received: when a job is completed, the next request processed is the one that has waited the longest.

Error messages or output produced by the PostScript **print** operator are sent back to the workstation over the same connection used for data. (The print operator is used to transfer messages to the host—it does not cause the LaserWriter to print on the page.) Data is carried transparently in both directions—no character codes are reserved for AppleTalk communication functions.

Data Transfer

Once a connection is opened, PAP's data transfer phase begins. In this phase, PAP has two functions: to transfer data over the connection, and to detect and terminate half-open connections.

PAP uses a "read-driven" data transfer model; before data can be sent, the other end of the connection must send a transaction request, thereby indicating that it's ready to read data. The PAP client at either end can request to receive data from the other end by issuing a PAPRead call. The other end can then write data through PAPWrite calls.

Closing the AppleTalk Connection

The AppleTalk protocols define an end-of-file indication. When the PostScript interpreter encounters that end-of-file indication, the LaserWriter sends a matching end-of-file indication back to the host, terminates the current job, and, if possible, starts a new one.

Typically, after the PAP client in the host has finished sending data to the LaserWriter and has received an EOF in return, it issues a PAPClose call to close the connection (see the PAPClose section, below).

Using PAP calls

Unless you are writing a remote AppleTalk-connected spooler, it is unlikely that you will use the PAP calls. They are provided here for those who need them. You should experiment with these calls only if you are the only user on the AppleTalk network: disrupting the normal packet exchange can terminate print jobs and even bring down the entire network. Table 3-2 lists the Pascal form for each PAP call together with the parameters the client must pass. These calls are then described in detail.

g4
Table 3-2
PAP Call Summary

FUNCTION SLInit	(PrinterName: Ptr; FlowQuantum: INTEGER): INTEGER;
FUNCTION GetNextJob	(VAR RefNum, CompState: INTEGER): INTEGER;
FUNCTION PAPOpen	(VAR RefNum: INTEGER; PrinterName: Ptr; FlowQuantum: INTEGER; StatusBuff: Ptr; VAR CompState: INTEGER): INTEGER;
FUNCTION PAPClose	(RefNum: INTEGER): INTEGER;
FUNCTION PAPRead	(RefNum: INTEGER; ReadBuff: Ptr; VAR DataSize: INTEGER; VAR EOF: INTEGER; VAR CompState: INTEGER):INTEGER;
FUNCTION PAPWrite	(RefNum: INTEGER; DataBuff: Ptr; DataSize: INTEGER; EOF: INTEGER; VAR CompState: INTEGER):INTEGER;
FUNCTION PAPStatus	(PrinterName: Ptr; StatusBuff: Ptr): INTEGER;
FUNCTION PAPRegName	(PrinterName: Ptr): INTEGER;
FUNCTION PAPRemName	(PrinterName: Ptr): INTEGER;
FUNCTION PAPUnload:	INTEGER

SLInit

When the LaserWriter is first started, it goes through its internal initialization and then issues an SLInit call to its PAP client. SLInit is used only by the LaserWriter. Its form is as follows:

FUNCTION SLInit (PrinterName: Ptr; FlowQuantum: INTEGER): INTEGER;

where:

PrinterName is the name of the LaserWriter (see Naming the LaserWriter, below);

FlowQuantum is an integer specifying the flow quantum equal to the number of 512 byte buffers (e.g., if FlowQuantum = N, then the flow quantum = 512*N bytes; the LaserWriter uses N = 8).

This call causes the PAP to do the following:

1. Open a service listener (SL) socket in the LaserWriter (by calling ATP to open a responding socket.)
2. Register the LaserWriter's name(s) in the LaserWriter's names table and bind them to the SL socket (by calling NBP).
3. Issue an ATPGetRequest call on this socket (so that the LaserWriter can respond to PAPOpen or PAPStatus request packets).

SLInit is executed synchronously.

As mentioned above, PAP uses both NBP and ATP. The use of NBP is strictly for the purpose of registering the LaserWriter's SL socket and for determining the address of a LaserWriter's SL socket when given its name.

GetNextJob

Just after the SLInit call completes, the PAP client in the LaserWriter makes a GetNextJob call to indicate that the LaserWriter is ready to accept jobs. GetNextJob is used only by the LaserWriter, not by the host. Its form is as follows:

FUNCTION GetNextJob (VAR RefNum, CompState: INTEGER): INTEGER;

where:

RefNum is a variable in which a reference number is returned when a connection has been opened.

This call puts the LaserWriter in the IDLE state, ready to accept connection requests. GetNextJob is also called after a printing job has been completed and the LaserWriter is again ready to accept another connection.

PAPOpen

This call is issued by a PAP client in a host to open a connection to a specified LaserWriter. The form of this call is as follows:

```
FUNCTION PAPOpen (VAR RefNum: INTEGER; PrinterName: Ptr; FlowQuantum:
                  INTEGER; StatusBuff: Ptr; VAR CompState: INTEGER ): INTEGER;
```

where:

RefNum	is the connection reference number returned after the connection has been opened.
PrinterName	is a pointer to the LaserWriter's entity name. An entity name consists of: the object name length byte, the object name, the type length byte, the type, the zone length byte and the zone. (For more information, see "Calling the AppleTalk Manager from Assembly Language" in the "AppleTalk Manager" chapter of <i>Inside Macintosh</i>)
FlowQuantum	is an integer specifying the flow quantum equal to the number of 512 byte buffers (e.g. if FlowQuantum = N, then the flow quantum = 512*N bytes; the LaserWriter uses N = 8);
StatusBuff	is a pointer to the buffer structure (given below) in which the printer status is returned to the caller during the opening process;
CompState	is an integer that can be monitored by the caller for call completion and error reporting. While the call is executing, this variable will have a value greater than zero. When the call has completed, it will assume either a value of zero (no error) or a negative value which is an error code.

PAPOpen causes the workstation's PAP code to do the following:

- Obtain the complete internet address of the LaserWriter's SL socket by issuing an NBP Lookup call.
- Open an ATP responding socket Rw
- Generate an 8-bit connection identifier (ConnID)
- Send a transaction request (TReq) to the LaserWriter's SL socket. This TReq has a PAP-type of OpenConn (see "PAP Packet Formats" in this chapter). This packet contains the ConnID, the address of socket Rw, the flow quantum for the workstation, and a wait time used by the LaserWriter for arbitration.

All packets related to a connection (sent by either end) must contain the correct connection identifier.

Since PAP uses ATP to transfer data, each of the two communicating PAP clients must discover the address of the ATP responding socket of the other connection end. Also, the amount of data transferred in an ATP transaction cannot exceed the size of the available receiving buffers at the end that issues the read requests. This maximum size (called the flow quantum) is sent by each end to the other when the connection is opened.

The structure of the `StatusRec` pointed to by `StatusBuff` is given by the Pascal type declaration:

```
TYPE StatusRec =          PACKED RECORD
                           SystemStuff: LongInt; {PAP internal use}
                           StatusStr: STR255      {status string}
                           END;
```

The caller must clear `StatusStr` before making the call. While the call is being processed, the caller can monitor `StatusStr`, in which PAP will continuously insert the status information being returned from the LaserWriter in PAP `OpenConnReply` packets. (see PAP packet formats, below). The possible result codes are 0, for connection accepted, and \$FFFF, for printer busy. The PAP client in the host might wish to display this string to provide appropriate feedback to the user.

`PAPOpen` is executed asynchronously. As soon as control returns to the caller (if the function's returned value equals `NoErr`) then the caller can monitor for call completion by examining the variable `CompState`.

Processing OpenConn requests: When the LaserWriter receives an ATP `TReq` of PAP-type `OpenConn`, its PAP client does the following:

If the LaserWriter is `BUSY` (processing a job), then it responds to the `OpenConn` with an ATP response of PAP-type `OpenConnReply` (2) indicating a server busy state (\$FFFF). (There is also a separate status request packet that yields the same information.) The status response packet travels over a path that's logically separate from the one through which the server is receiving its current job.

At the workstation end, if an `OpenConnReply` of `BUSY` (\$FFFF) is received, then the workstation's PAP waits approximately two seconds and issues another connection request. Each time it repeats this request, it updates its `waitTime` (the current value of this wait time is sent with each `OpenConn`). Each of these `OpenConn` requests is issued with a retry count of 5 and retry interval of about 2 seconds. If the LaserWriter is inoperative, or in its 6-

second imaging loop, it won't be able to respond; then the transaction will terminate without receiving a reply at all. The workstation's PAP updates the wait time and tries again.

If the LaserWriter is IDLE, then it reacts to the OpenConn by going into an arbitration (ARB) state for approximately two seconds. In the ARB state, the PAP receives all incoming OpenConn requests and selects the one from the workstation that has waited the longest. The time, in seconds, that a workstation has been trying to open a connection (the WaitTime) is sent with the OpenConn. After becoming IDLE, the LaserWriter loads the WaitTime of the first OpenConn that it receives into a variable called OldestReq. It then compares the WaitTime of each subsequent OpenConn request with OldestReq. If the LaserWriter receives a request that has been waiting longer than the pending request, it makes that request the pending request, and saves its WaitTime in OldestReq. Otherwise, it responds to the new request with an OpenConnReply, indicating BUSY (\$FFFF). At the end of the ARB interval, PAP opens an ATP responding socket Rs, and sends an ATP response of PAP-type OpenConnReply indicating "Connection accepted" (0) to the selected request. This carries the ConnID received in the OpenConn, the address of socket Rs and the flow quantum of the LaserWriter end (set by the SLInit call—it is currently 8 for the LaserWriter). The connection is then open; the workstation's job is processed, and the LaserWriter is in the BUSY state.

PAPRead

This call is issued by the PAP client at either end of the channel to read data from the other end over the connection specified by the reference number.

```
FUNCTION PAPRead (RefNum: INTEGER; ReadBuff: Ptr; VAR DataSize: INTEGER; VAR  
EOF: INTEGER; VAR CompState: INTEGER):INTEGER;
```

where:

RefNum	is the connection reference number (see PAPOpen);
ReadBuff	is a pointer to the buffer into which the data is to be read;
DataSize	is an integer in which the number of bytes of data read into the buffer is returned when the call completes;

EOF is an integer in which the end-of-file indication received from the other end is returned to the caller (a non-zero value indicates an end of file; otherwise a value of 0 is returned);

CompState is an integer that can be monitored by the caller for call completion and error reporting (see PAPOpen).

PAPRead does the following:

1. Provides PAP with a read buffer into which the data will be read. (Note that PAP assumes that the buffer to which ReadBuff points is no smaller than this end's flow quantum specified in the PAPOpen call.)
2. Calls ATP to send an ATP transaction request with PAP-type SendData and an ATP bitmap reflecting the size of the call's read buffer. This transaction has a retry count of "infinite" (255) and a retry interval of 15 seconds. To prevent duplicate delivery of data to PAP's clients, these ATP transactions use ATP's exactly-once mode.

The receipt of an ATP TReq packet with PAP-type SendData means that a PAPRead is pending at the other end. This "send credit" is remembered by the PAP code, and used to service any pending or future PAPWrite calls issued by its client.

PAPRead is executed asynchronously. As soon as control returns to the caller (if the function's returned value equals NoErr) then the caller can monitor for call completion by examining the variable CompState .

When the call has completed without error, then the variable DataSize is equal to the number of bytes of data received into the buffer. (If the call completes with an error, the value of this variable is unpredictable.)

PAPWrite

This call is issued by the PAP client to write data to the other end of the connection specified by the reference number.

```
FUNCTION PAPWrite (RefNum: INTEGER; DataBuff: Ptr; DataSize: INTEGER; EOF:
INTEGER; VAR CompState: INTEGER):INTEGER;
```

where:

DataBuff is a pointer to the data to be written;

DataSize is equal to the number of bytes of data to be written; if the data size exceeds the flow quantum of the other end (specified in the PAPOpen call) PAPWrite will return with an error;

EOF is the end-of-file indication to be sent to the other end (a non-zero value indicates end of file; otherwise a value of zero should be sent).

When a PAP client (at either end) issues a PAPWrite call, PAP examines its internal data structures to see if it has received a "send credit". If it has, then it takes the data from the PAPWrite and sends it in ATP Response packets with PAP-type Data. The EOM bit is set in the last of these ATP response packets. If no send credit has been received, then PAP queues the PAPWrite call and awaits a SendData from the other end.

When a PAP client issues the last PAPWrite call for a job, it must send an End-of-File indication with that call's data. The EOF indication is delivered to the PAP client at the other end. For this purpose the client can issue a PAPWrite call with no data; in this case, just an EOF indication is conveyed to the client at the other end.

PAPWrite is executed asynchronously. As soon as control returns to the caller (if the function's returned value equals NoErr) then the caller can monitor for call completion by examining CompState.

PAPClose

When the PAP client at either end issues a PAPClose call, PAP closes the connection. The form of this call is as follows:

```
FUNCTION PAPClose (RefNum: INTEGER) : INTEGER;
```

PAPClose closes the connection specified by RefNum. It cancels any pending PAPRead and PAPWrite calls for the indicated connection.

PAPClose is executed synchronously. It sends an ATP transaction request of PAP-type CloseConn to the other end. The end receiving the CloseConn should immediately send back, as a courtesy, an ATP transaction response of PAP-type CloseConnReply. To close a connection's end, it is important to cancel any pending ATP transactions issued by that end, including

the Tickle transaction. Note that the end receiving the CloseConn might not cancel these pending transactions immediately, as it will probably be at interrupt level.

At the LaserWriter end, receipt of the CloseConn causes the connection to be closed, but the LaserWriter will continue in the BUSY state until it actually finishes processing the data for the job. When it finishes, the PAP client in the LaserWriter issues a GetNextJob call. This call puts the LaserWriter back in the IDLE state, ready to accept further connection requests.

PAPUnload

PAPUnload completely closes down PAP.

FUNCTION PAPUnload: INTEGER

This call can be used at either end to cause the PAP data structures to be unloaded and currently open connection(s) to be closed. It could be used on the LaserWriter, for instance, if the mode switch is moved from AppleTalk to one of the serial communication settings. In the workstation, the client would use this before exiting to the Finder.

PAPStatus

The PAP client in the workstation can issue a PAPStatus call to find out the status of the LaserWriter at any time—even if the PAP client hasn't opened a connection to the LaserWriter.

FUNCTION PAPStatus (PrinterName: Ptr; StatusBuff: Ptr): INTEGER;

where:

PrinterName	is a pointer to the entity name (see PAPOpen) of the LaserWriter whose status is to be determined;
StatusBuff	points to a structure of type StatusRec (see PAPOpen).

PAPStatus is executed synchronously, and upon completion returns a string with the status message sent by the LaserWriter.

PAPRegName

This call is used by the LaserWriter only. It registers a name (as the entity name for the print server) on the LaserWriter's listening socket.

```
FUNCTION PAPRegName (PrinterName: Ptr): INTEGER;
```

where:

PrinterName points to a structure of type EntityName.

PAPRemName

This call is used by the LaserWriter only. It deregisters a name from the LaserWriter's listening socket.

```
FUNCTION PAPRemName (PrinterName: Ptr): INTEGER;
```

Naming a LaserWriter

A LaserWriter is identified by a three-part name constructed according to the Name Binding Protocol. The first (or object) part (the printer's individual name) is initially "LaserWriter" but may be set to any other value by means of PostScript's `setprintername` operator. The second (or type) part is *always* "LaserWriter", and the third (or zone) part is unspecified. You can connect more than one LaserWriter to the same AppleTalk network—if you add a LaserWriter with the same name as an existing one, it will automatically choose a new name ("LaserWriter1", "LaserWriter2", etc.). The PAP client in the LaserWriter can use the calls `PAPRegName` and `PAPRemName` to register and remove (deregister), respectively, a LaserWriter's name.

PAP Packet Formats

Packets sent by ATP in response to PAP calls include a PAP header. This is built using the user bytes of the ATP header, and in some cases by sending four or more bytes of PAP header in the data part of the ATP packet. In all cases, the first of the ATP user bytes is the ConnID, and the second the PAP-type of the packet:

The permissible PAP-type field values are:

- 1 OpenConn
- 2 OpenConnReply
- 3 SendData
- 4 Data
- 5 Tickle
- 6 CloseConn
- 7 CloseConnReply
- 8 SendStatus
- 9 StatusReply

For data packets, the third ATP user byte is the EOF indication. Figure 4-2 shows the PAP header format for different PAP packets.

4

Detecting Half-open Connections

A half-open connection exists when one of the connection ends terminates the connection without informing the other end. Half-open connections must be detected and closed to ensure the integrity of the network.

For this purpose, PAP maintains a connection timeout of approximately two minutes at each end. This timer is started as soon as the connection is opened. Whenever a packet of any sort is received from the other end of the connection, the timer is reset. If the timer expires, the connection is torn down—it's assumed that the other end has "died" or closed its connection.

For this mechanism to work properly, PAP exchanges control packets to signal that the connection ends are alive, even though no data is being exchanged on the connection. This process is called tickling and the control packets are called tickling packets. As soon as a connection is opened, each end starts an ATP transaction of PAP-type Tickle. This Tickle transaction has a retry count of infinity (ATP uses a value of 255 to signify infinite retries), and a retry interval of half the connection timeout. Tickle packets are sent to the other end's ATP responding socket (Rs or Rw). The receiver of a Tickle packet must reset its connection timer but must not send a transaction response. Each end cancels Tickle transactions when the connection is closed.

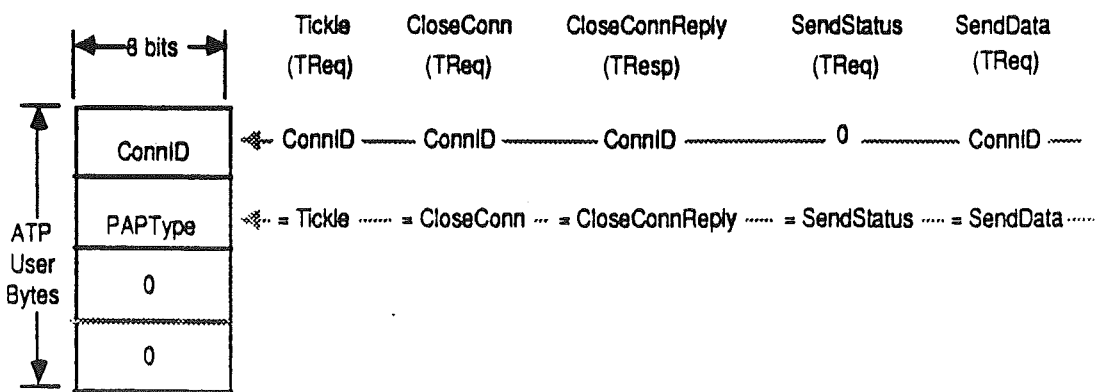
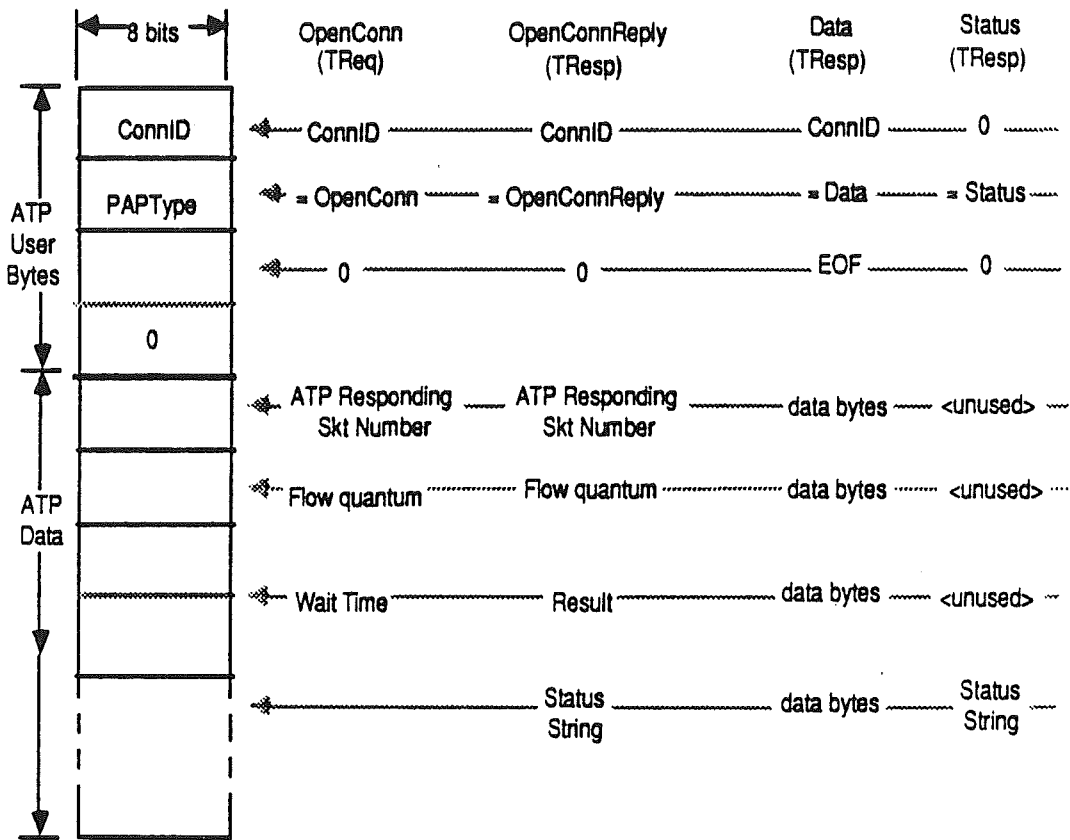


FIG 4-2 KROST Header For 2

89 4-15.1

In interactive mode, a job consists of a dialog in which you issue a PostScript command and the LaserWriter generates a response and prompts for the next command. This is useful for learning about PostScript and testing commands on the LaserWriter.

Accessing the LaserWriter directly

You can connect a Macintosh running MacTerminal directly to the port on the back of the LaserWriter. You can also connect an ASCII terminal or another computer running terminal emulation software. This approach enables you to access the LaserWriter without having to go through several levels of software. You can do your programming either in interactive mode or in batch mode. In interactive mode, you can hold a dialog with the LaserWriter, instructing it to execute PostScript code that you send to it. This can be a convenient way to test pieces of your PostScript programs. In batch mode, you send an entire file at a time for the LaserWriter to execute.

You connect a Macintosh (or an Apple // running Access //) to the LaserWriter's 9-pin connector using an Apple Modem cable or ImageWriter cable. (These cables must have the 9-pin connector; the 8-pin connector will not work.) You can also connect any terminal (or computer running terminal emulation software) with a standard RS232C interface directly to the LaserWriter, usually via the 25-pin connector. When making this connection, you generally need to use a "null modem" or "modem eliminator" device that reverses the Transmit Data and Receive Data signals. (See Appendix A for serial interface signals.)

Working interactively

There are two ways to put the LaserWriter into interactive mode. The first is to select one of the batch mode switch positions (1200 or 9600), make sure the attached terminal is set to the correct baud rate and parity, and invoke the PostScript procedure `executive` (type "executive" followed by return or new-line). The other way is to redefine the meaning of the "Special" switch position (by changing some PostScript parameters in the LaserWriter's RAM) so that selecting the Special setting invokes interactive mode instead of emulation mode.

In interactive mode, the state of PostScript's virtual memory persists until the job is ended by your explicit request. While you're typing, the LaserWriter echoes the characters you type back to your terminal (so you can see them). You can use the following special characters for editing while you type:

- Backspace (Control-H) erases one character
- Delete (Rubout) same as backspace.
- Control-U Erases the current line.
- Control-R Re-displays the current line.
- Control-C Aborts the entire statement and starts over.

Interactive mode continues until you type Control-D (the serial end-of-file character), execute a PostScript quit command, or change the mode switch setting.

❖ *Note:* On a Macintosh the Command key is the equivalent of the control key on other computers.

Below a short series of exercises is presented for interacting with the PostScript interpreter directly, using MacTerminal as an ASCII terminal interface. Although this example uses a Macintosh running MacTerminal, it applies, with slight modifications, to any ASCII terminal or computer running terminal emulation software.

1. Start out with the printer OFF, with nothing attached to any of the connector ports, and with the mode switch on the back of the Laserwriter in the 1200 baud position.
2. Cable the Macintosh to the LaserWriter using either a 9-pin to 9-pin cable or a 9-pin to 25-pin cable (a standard ImageWriter cable will work). Connect one end of the cable to the appropriate port on the printer (9- or 25-pin) and the other end to the Macintosh's 9-pin MODEM port.
3. Turn the printer ON. Several things will happen. First, the green light will blink. This is the printer's normal warm-up indicator. Next, the yellow light will blink. This is the printer's signal that data is being processed. In this case the printer is processing the start-up page. Within two minutes the test page should be printed.
4. Start up a disk with MacTerminal on it. For these examples you should also have MacWrite handy.
5. Check to see that the following settings are correct in the "Settings" menu:

Terminal: VT100; ANSI; Underline; U.S.; 80
column; On-Line; Auto Repeat; Auto
Wraparound

Compatibility: Baud Rate = 1200; 8 bits, Parity = none;
Handshake = none; connection = another
computer; connection port = modem

File Transfer: Transfer Method = text

6. Now press Command-T. (The Command key is the one immediately to the left of the space bar.) This causes printer status to be displayed on the screen. It should look something like this:

```
%% [status: waiting; source serial 9] %%
```

7. Press Command-D (this will stabilize the printer). If you now press Command-T again you should get

```
%%[status: idle]%%
```

8. At this point you're ready to 'talk' to PostScript. To enter interactive PostScript mode, type the word `executive` followed by a carriage return. (Since the LaserWriter is still in batch mode, the characters you type aren't echoed back to you.)

9. The following should appear:

```
PostScript (tm) version 23.0  
Copyright (c) 1984 Adobe Systems Incorporated.  
PS>
```

PS> is the PostScript prompt. This means that you're now 'talking' directly to the PostScript interpreter. Each time the LaserWriter displays the "PS>" prompt, it is waiting for you to type in a PostScript statement followed by return or new-line character. It then executes that statement and displays another "PS>" prompt.

10. Test to see if the connection is working by typing `showpage` and pressing return. This should eject a blank piece of paper.

11. Now try printing something simple. In response to the PS> prompt, type the following commands, following each with a carriage return.

```
/Times-BoldItalic findfont  
72 scalefont setfont  
100 100 moveto  
30 rotate  
(Put your name here) show  
showpage
```

Note that you don't get a prompt right away after the show statement. This is because the Postscript interpreter is busy creating a scaled and rotated font.

This exercise should have created a page with your name printed at an angle.

Working in batch mode

To enter batch mode, type Command-D. Batch mode lets you stream many lines of PostScript to the printer at once. (This can be done in interactive mode as well.) MacTerminal echoes all lines of text back to the screen in Interactive mode. In batch mode, the file is simply sent to the printer and executed. To test out this mode follow these steps:

1. Create a PostScript file using MacWrite or another text editor. Make sure you save the file using the SAVE AS command and selecting the TEXT (ASCII) format for saving the file.

```
2 2 scale
/Times-BoldItalic find font 27 scalefont setfont
/rays
{0 1.5 179
  {gsave
   rotate
   0 0 moveto 108 0 lineto
   stroke
   grestore
  } for
} def
125 200 translate
.25 setlinewidth
newpath
0 0 moveto
(StarLines) true
charpath clip
newpath
54 -15 translate
rays
showpage
```

2. Write the above text (in TEXT format) to your disk as "PSTEST".

3. Go back to MacTerminal (make sure your settings are correct, as given in step 5 of the previous example). Select "Send File" from the File menu, and then open the PSTEST file you just created. This procedure should send the entire file to the printer. If you are still in interactive mode you will see the ASCII playback on the screen, if you entered Control-D (to return to batch mode), you will not. This file may take a little while to execute, but it's worth the wait. In about two or three minutes the page should print.
 4. You can now experiment with the switch settings. Turn the printer off and switch it to 9600 baud. Turn it back on and set the communication settings in MacTerminal to 9600 and try these exercises again.
- ❖ *Note:* Normal Macintosh applications that are supported by the LaserWriter Print Manager use the AppleTalk connector. This should not be used with any other communication link to the printer (don't try to hook up AppleTalk and RS232 at the same time). The AppleTalk cable connects to the 9-pin port on the printer and the PRINTER port on the back of the Mac. For further information, see Appendix A.

Using the Diablo 630 emulator

In Diablo 630 emulation mode, the LaserWriter interprets incoming data as text and Diablo 630 control codes rather than as a PostScript program. This capability is intended for printing simple text files in this popular format, primarily output from software packages that don't support PostScript.

Invoking the Diablo emulator

To invoke the Diablo emulator, set the server mode switch to the "Special" position and connect one of the LaserWriter's serial ports to the host's RS232C interface. Text to be printed can then be sent at 9600 baud with any parity.

Most of the information about serial communication in Appendix A applies for Diablo emulation. However, different meanings are given to control characters—all characters are treated according to the Diablo 630 protocol as shown in ~~table 4-3~~. The LaserWriter still sends XON and XOFF characters to control the flow of data from the host.

Figure 4-3

<u>Margins & Formatting</u>		
Set Top Page Margin (at current position). ***	ESC	T
Set Left Margin (at current position). ***	ESC	9
Set Horizontal Tab (HT) Stop (at current position). ***	ESC	1
Set Right Margin (at current position). ***	ESC	0
Set Vertical Tab (VT) Stop (at current position). ***	ESC	-
Set Lines Per Page to (n). **	ESC	FF (n)
Set Bottom Page Margin (at current position). ***	ESC	L
Clear Top and Bottom Margins	ESC	C
Clear Horizontal Tab (HT) Stop (at current position). ***	ESC	8
Clear all HT and VT Stops	ESC	2
Set Horizontal Motion Index (HMI)* to (n - 1). **	ESC	US (n)
Set Vertical Motion Index (VMI)* to (n - 1). **	ESC	HS (n)
Return HMI Control to <u>Spacing</u> Switch.	ESC	S
<u>Carriage Movement</u>		
Absolute HT to print column (n). **	ESC	HT (n)
Enable Auto Backward Printing	ESC	/
Disable Auto Backward Printing	ESC	\
Reverse Printing Mode	ESC	,
Normal Printing Mode	ESC	.
Forward Print Mode ON	ESC	3
Backward Print Mode ON - Forward Mode OFF (clear with CR)	ESC	6
<u>Paper Movement</u>		
Absolute VT to line (n). **	ESC	VT (n)
Perform Negative Line Feed	ESC	LF
Perform Half-Line Feed	ESC	U
Perform Negative Half-Line Feed	ESC	D
<u>Printing</u>		
Graphics Mode ON (clear with CR)	ESC	3
Graphics Mode OFF	ESC	4
HyPlot ON - Absolute Move (clear with CR)	ESC	G
HyPlot ON - Absolute Plot (clear with CR)	ESC	G BEL
HyPlot ON - Relative Move (clear with CR)	ESC	V
HyPlot ON - Relative Plot (clear with CR)	ESC	V BEL
Change Plot Character to (character)	ESC	. (new character)
Set Plot Precision *	ESC	, hv
Print in Secondary Color (red)	ESC	A
Print in Primary Color (black)	ESC	B
Print Suppression ON (clear with CR)	ESC	7
<u>Word Processing Commands</u>		
Proportional Space ON (clear with ESC S)	ESC	P
Proportional Space OFF	ESC	Q
Offset Selection *	ESC	DC1 (n)
Auto Underscore ON	ESC	E
Auto Underscore OFF	ESC	R
Bold Print ON (clear with CR)	ESC	O
Shadow Print ON (clear with CR)	ESC	W
Bold/Shadow Print OFF	ESC	&
Increase Carriage Settling Time to 20ms (clear w/ESC N)	ESC	%
Backspace 1/120*	ESC	BS
Program Mode ON	ESC	SO M
Cancel all WP modes except Prop Space & Car Settling Time	ESC	X
Auto Center ON (clear with CR)	ESC	=
Auto Justify ON	ESC	M
Margin Control ON (overrides MARG CONT key)	ESC	\$
Margin Control Mode controlled by MARG CONT key	ESC	*
Clear Increased Carriage Settling Time	ESC	N
<u>Miscellaneous Commands</u>		
Initiate Remote RESET	ESC	CR P
Print Print Wheel character HEX 20	ESC	Y
Print Print Wheel character HEX 7F	ESC	Z
Enter Program "Here is . . ." Mode	ESC	(
Exit Program "Here is . . ." Mode	ESC)

Figure 4-3 Diablo Control Characters

❖ *Note:* Not all print drivers in microcomputer operating systems support the XON/XOFF protocol ; it may be necessary to obtain a separate software package to support this protocol.

Changing print parameters

All the parameter settings that can be changed with Diablo software commands are initialized in the emulator as they are in the Diablo 630 printer. Other parameters require setting hardware switches or changing print wheels in the Diablo; in the LaserWriter these are system parameters that can be changed via PostScript commands. The complete set of persistent parameters pertaining to Diablo emulation is given in table 4-~~x~~³.

Table 4-~~x~~³
Persistent parameters for Diablo emulation mode

Parameter	Initial setting
pitch	10
font	Courier
font for bold	Courier-Bold
auto-linefeed	off

The Diablo emulator supports all the standard LaserWriter typefaces. The default font is Courier, which is the fixed-pitch font most commonly used on daisywheel printers, and which is most likely to give correct results for typical microcomputer application programs. Note that the plain and bold fonts are specified separately. Thus, one could use Courier for regular printing and Courier-Oblique for bold; then the "bold" text would print as italic instead.

New persistent parameters that control Diablo emulation and mode selection via the "Special" switch have been assigned using the cells accessed by the PostScript commands `eescratch` and `seteescratch`.

The `eescratch` locations shown in table 4-~~x~~⁴ have been assigned. Thus, for example, to change the meaning of the "Special" switch position from Diablo emulation to PostScript interactive mode, issue the command:

58 1 seteescratch

The default value of every eescratch cell is zero.

Table 4-X⁴
eescratch location assignments

Location	Assignment
58	selects the function of the "Special" switch setting: 0 means Diablo 630 emulation mode; 1 means PostScript interactive mode; and other values are reserved for future capabilities.
59	the value 1 enables the Diablo auto-linefeed feature.
60	selects the Diablo pitch number of characters per inch . Reasonable values are 10, 12, and 15; 0 selects 10.
61	a font number that forthe "bold" font to be used for Diablo emulation. If the number is 0 (selecting Courier) then 1 (selecting Courier-Bold) is used instead. To actually select Courier as the "bold" font, use some illegal font number such as 255.
62	selects the plain font used for Diablo emulation. This is a font number taken from . The default value (0) selects Courier.
63	used internally

The EEROM in which the persistent parameters are stored can be written only a limited number of times before wearing out. Each location in the EEROM is capable of approximately 10,000 writes. For this reason, the EEROM is used only for parameters that are expected to change infrequently. The copy count is an exception; it is implemented in such a way that the wear is distributed over a large number of locations.

At power-on time, the contents of the EEROM are checked for consistency, and an entry named `eerom` in `statusdict` is used to report the result. Normally, `eerom` contains `true`. If an inconsistency is detected, `eerom` is redefined to be a 512-character PostScript string into which are read the entire contents of the EEROM; then the page count is set to zero and all parameters are reset to default values. If the EEROM fails altogether, `eerom` is set to `false` and the software shifts to a simulation of the EEROM parameters in RAM; all the operations for setting and reading parameters continue to work, but the values no longer survive across power-off.

Deviation From Diablo Protocol

The LaserWriter emulates the Diablo as closely as possible; however, there are some differences of which you should be aware:

Detecting the End of a Document

The LaserWriter can only detect the end of a document by noticing that data has stopped arriving. After the last page is processed, all Diablo settings such as margins, tabs, and spacing stay in effect for about 30 seconds (or another period to which the default wait timeout has been changed). Then a Diablo "reset" operation is automatically performed to restore all settings to standard values; i.e., the margins are cleared, spacing is put back to standard, and tab settings and any special word processing modes are cleared. The LaserWriter actually prints a page when it either reaches the bottom of the page or receives a form-feed (Command-L) character. If the last page of a document isn't full and doesn't end with a form-feed, it won't be printed immediately. Instead, it will be printed when the LaserWriter resets approximately 30 seconds later, or as part of the next document (at the top of the first page). When documents are being printed in close succession, make sure that each one has a final form-feed so that they are not run together.

Bold and Double-strike

Some text processors produce a bold style by double-striking a character. That will not appear as bold in the LaserWriter. Only the bold produced by issuing the proper Diablo command sequence (escape-O) will result in bold characters.

Proportional Fonts

Times-Roman and Helvetica are narrow fonts that may look squeezed if no adjustment of page width is made by the word processor. Few non-Macintosh text processing programs are capable of producing correctly formatted output using proportionally spaced fonts such as these.

Paper Positioning

The emulator uses exact positioning on the paper. Output from a word processor that has attempted to compensate for slippage on vertical movement may appear slightly uneven.

Unsupported Commands

The following Diablo 630 commands are not supported by the LaserWriter:

- print suppression
- HY-Plot
- extended character set
- the ability to download information for print wheels, including program mode
- the ability to override printwheel spacing (for proportional spacing), although the offset for proportional spacing can be changed
- page lengths other than 11 inches
- paper feeder control
- hammer energy control
- remote diagnostic
- backward printing control (note, however, that "reverse printing" is supported)

MS-DOS Communication Parameters

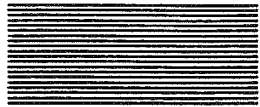
If you are using an IBM-PC or compatible computer, you can issue the following MS-DOS commands to set up serial port 1 for communication with the LaserWriter:

```
MODE COM1:9600,n,8,1
```

```
MODE LPT1:=COM1:
```

These commands set the baud rate to 9600 and map printer output to the serial port. This by itself is not sufficient to support XON/XOFF flow control—some applications may handle this protocol themselves; otherwise a different printer driver should be

installed to avoid communication problems while printing large documents.



Appendix A



Serial Data Communication

Using the serial communication channels

The LaserWriter has two serial channels, one wired to a DB-9 (RS422) connector and the other to a DB-25 (RS232C) connector (see figure A-1). Either channel can be used for conventional asynchronous serial communication. (The 9-pin connector is also used (in conjunction with the mode switch) for connecting to AppleTalk, but serial and AppleTalk communication are incompatible and will never occur at the same time.)

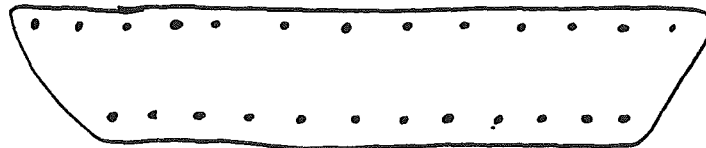
When the LaserWriter is in any of the serial I/O modes, it uses one of the two channels to send and receive serial data encoded in ASCII. Certain character codes serve special purposes, such as Command-D to mark end-of-file. At the beginning of a job, both channels are enabled with independent baud rate and parity. The first channel to receive a character is chosen to execute the job. (The other channel is not disabled; if characters start to arrive on it, they are buffered and that channel is selected when the current job is finished.) When the end-of-file character is received and the program terminates, the LaserWriter sends an end-of-file character, ends the job, and (if possible) starts a new one.

Using the RS422 port

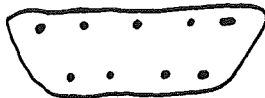
The pins for the serial RS422 port are assigned as shown in table A-1.

Table A-1
RS422 port pin assignments

Pin	Signal
1, 3	Signal Ground
4	Transmit Data +
5	Transmit Data -
8	Receive Data +



DB-25



DB-9

Fig A-1. Serial Port Connectors

This pin assignment is compatible with the Macintosh. You can connect a Macintosh directly to a LaserWriter using an Apple Modem cable and communicate with it using MacTerminal.

Using the RS232C port

The pin assignments for the 25-pin RS232C port are shown in table A-2.

Table A-2
RS232C Pin Assignments

Pin	Signal
2	Transmit Data
3	Receive Data
4	Request To Send (optional; needed only if host requires it)
7	Signal Ground
20	Data Terminal Ready (optional; needed only if host requires it)

The other signals are not used.

Technically, the LaserWriter is configured as a DTE. This means you can directly connect it to a modem or to a host computer configured as a DCE device without any signal reversals. Connecting to another DTE device requires interposing a "null modem", which, at a minimum, involves reversing the Transmit Data and Receive Data signals.

Changing communication parameters

On the LaserWriter, serial communication is always asynchronous, start-stop, with 8 data bits per character (the high-order bit may or may not be used for parity), one start bit, and two stop bits. There are three programmable parameters: *channel* (9- or 25-pin connector), *baud rate*, and *parity*. Switch setting "1200" establishes communication with standard parameters (1200 baud, parity ignored). The "9600" and "Special" switch settings use

whatever parameters have been set previously—if no parameters have been set, the defaults are 9600 baud, parity ignored. If you're trying to make contact with a LaserWriter for the first time, and you don't know which parameters may have been set previously, you should start with the "1200" setting.

You can change the channel, baud rate and parity with the PostScript `statusdict` operators `setscbatch` (for the "9600" setting) and `setscinteractive` (for "Special"). (SCC stands for Serial Communications Controller, which is the device that operates the two I/O connectors.)

setscbatch operator

The PostScript `setscbatch` operator determines how serial communication is performed on the specified channel for subsequent batch jobs when the switch is in the "9600" position. As arguments, it takes three integers designating channel, baud rate, and parity:

```
channel baud parity setscbatch
```

In a PostScript program, these parameters are specified as follows:

channel	The 9- and 25-pin connectors are designated by the integers 9 and 25.
baud rate	Given as an integer, such as 1200 or 9600. (The maximum baud rate supported by the software is 9600.)
parity	Specified by an integer from 0 to 3: 0 Ignore— the high-order bit of each 8-bit character received is ignored, and the high-order bit of each character transmitted is zero. 1 Odd— the high-order bit of each 8-bit character received is checked for odd parity (a PostScript <code>ioerror</code> occurs if it is incorrect), and each character transmitted has odd parity. 2 Even— like odd, but for even parity. 3 None— all 8 bits of each character are treated as data, and no checking is performed.

Note that the baud rate and parity may be set independently for each of the two channels. The new baud rate and parity do not take effect until the end of the current job. Setting the baud rate to zero disables the channel, but disabling both channels is not permitted.

`scinteractive` is identical to `setscbatch`, but sets serial communication parameters to be used when the switch is in the "Special" position (which selects either interactive or emulation mode).

To determine the currently selected values for these parameters, use the operators `scbatch` and `scinteractive`:

`scbatch` takes a channel number (9 or 25) and returns the "9600" (batch) baud rate and parity previously set for that channel:

```
channel scbatch baud parity
```

The default is 9600 baud, parity ignored (0).

setscinteractive operator

The `scinteractive` operator is identical to `scbatch`, except that it returns the "Special" (interactive or emulation mode) baud rate and parity for the specified channel (the default settings are 300 and 0).

Controlling communication

The serial communication protocol is quite minimal. There are several character codes reserved for communication functions and not passed through to PostScript:

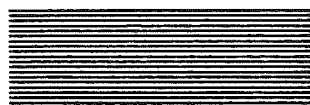
Control-C (\$03)	Interrupt—causes a PostScript interrupt operator to be executed. (See the <i>PostScript Language Manual</i> .)
Control-D (\$04)	End-of-file
Control-Q (\$11)	XON—start output
Control-S (\$13)	XOFF—stop output

Control-T (\$14)	Status query—when received over either channel, Control-T elicits a one-line status message over the same channel. This channel need not be the one through which the LaserWriter is receiving its current job. (Status messages are described in Chapter 1).
Return (\$0D)	End-of-line
Line-feed (\$0A)	End-of-line (ignored if it immediately follows a Return)

The LaserWriter uses XON/XOFF flow control and expects the host to do likewise. For batch mode operation, XON/XOFF flow control is required—note in particular that the RS232C Data Terminal Ready (DTR) signal for flow control isn't supported. Failure to conform to XON/XOFF flow control will result in an ioerror when transferring files longer than about 5000 characters.

There is no way to delimit the reserved control characters in order to pass them through as data to PostScript; nor is there any way to transmit characters in the high ASCII range (128 to 255) when the high-order bit is being ignored or used for parity. Thus, the serial link is not a fully transparent channel. However, this causes no difficulty in normal use because the PostScript language consists entirely of printable characters. The language itself provides a means for encoding arbitrary characters in strings (the “\ nnn” escape sequence). True binary data, such as images and encrypted programs, is transmitted in hexadecimal.

When a job terminates, the LaserWriter sends a Control-D end-of-file character over the serial channel. This allows the host application program to synchronize with the LaserWriter (if desired) and to correlate a given batch of output with the job that generated it. Note that the application program need not wait for one job to finish before beginning to send the input for the next job.



Appendix B



Apple Laser Prep dictionary

A LaserWriter dictionary is a PostScript data structure that links names to objects. A font is one example of a dictionary: it links the names of the fonts to the routines that are actually used to draw the fonts. The Laser Prep header file is also a PostScript dictionary. It contains the code to translate Macintosh operations into PostScript, and the keys by which those operations can be accessed.

Using the Laser Prep dictionary

The LaserWriter driver downloads the Laser Prep dictionary to the printer on a permanent basis whenever a document is to be printed and the dictionary is not available on the printer. Such functions as selecting fonts, drawing polygons, and managing document information are performed by the procedures defined in this dictionary. All characters in PostScript are printable characters. The full text of the dictionary procedures is printed here; some of the comments have been changed for readability.

Most of these procedures are defined into a global dictionary as they are encountered, while others are executed immediately and serve to set global state for later print jobs.

The names used herein are defined as PostScript names and should not be redefined or reused for other purposes by clients who are defining their own PostScript code. In most cases name conflicts are avoided by using names which are two and three characters in length; it is therefore usually safe to use names that are more than three characters.

Printer password

The LaserWriter password constant constitutes the first string in the LaserPrep string list resource. Clients who wish to change their password on the LaserWriter printer may make the header compatible by changing this string appropriately. Its length is 7 characters so that it may accommodate passwords up to 7 characters without need to change the length of the string or resource.

0000000

Printer initialization

The initialization process involves establishing the current dictionary, specifying the parameter settings for the communication port (9, for 9-pin), the baud rate (0, for the default 9600), and parity (3, for none). The wait timeout is set for 300 seconds.

```
serverdict begin exitserver
systemdict /statusdict known(statusdict begin 9 0 3 setsccinteractive
/waittimeout 300 def end)if
/md 200 dict def md begin
```

Dictionary version number

The dictionary version number must match the version number expected by the driver. The dictionary version number documented here is 13. That version number is specified here in PostScript.

```
/av 13 def
```

Leveltwohead

text

```
/mtx matrix currentmatrix def
/s30 30 string def
/s1 ( ) def
/pys 1 def
/pxs 1 def
/pyt 760 def
/pxt 29.52 def
/por true def
/xl(translate)def
/fp{pnsh 0 ne pnsv 0 ne and}def
```

Leveltwohead

The following array defines the QuickDraw grafVerbs in PostScript. In Pascal these correspond to:

```
GrafVerb = (frame, paint, erase, invert, fill);
```

The verbs are defined in the above order with a few additional verbs that are used internally.

◆ *Note:* The verb `invert` is not legal in PostScript; it is non-functional.

```
/vrb [  
{fp(gsave 1 setlinewidth pnsh pns scale stroke grestore)if newpath}  
{eofill}  
dup  
{newpath}  
2 index  
dup  
{initclip eoclip newpath}  
{}  
dup  
2 copy  
] def  
  
/xdf(exch def)def  
currentscreen /spf xdf /rot xdf /freq xdf
```

Execute specified QuickDraw verb

This procedure, called with the ordinal value of the QuickDraw verb on the PostScript stack, executes the appropriate verb procedure from the `/vrb` array above.

```
/doop {vrb exch get exec} def
```

Accept global values

This procedure accepts a portrait landscape flag, x and y translation coordinates, x and y scale factors, and x and y resolution parameters. It doesn't alter the current PostScript state yet.

```
/psu{2 index .72 mul exch div /pys xdf div .72 mul /pxs xdf /pyt xdf /pxt  
xdf /por xdf}def
```

Alter PostScript state

Here the actual PostScript state is altered according to global values set previously.

```

/txpose(dup 1680 eq userdict /note known({legal}{note}ifelse){pop}ifelse
dup 1212 eq(54 32.4 xl)if 1321 eq(8.64 -.6 xl)if
pxs pys scale pxt pyt xl por not(270 rotate)if 1 -1 scale)def

```

Provide oblique font

The procedure below is used to provide the slanted or obliques style (derived italic font) of a font. A point size and a flag are required.

```

/obl ({0.212557 mul}{pop 0} ifelse) def

```

Establish font

This routine establishes the font to be used in a subsequent operation. It requires three parameters: a font size, an oblique flag, and a font dictionary.

```

/sfd ([ps 0 ps 6 -1 roll obl ps neg 0 0] makefont dup setfont) def

```

Headings TBD

Leveltwohead

```

set font: obliqueflag /fontname

```

```

/fnt(findfont sfd)def

```

Leveltwohead

text

```

% test bit in an integer: integer bit-to-test

```

```

% returns flag and the original integer

```

```

%-----

```

```

/bt(1 index and 0 ne exch)def

```

```

%-----

```

```

% set style flags from the ordinal value of style

```

```
% style array contains flags for Bold, Italic, Underline, Outline, Shadow.  
% The ordinal value of style is also saved
```

```
-----  
-----
```

```
/sa 6 array def  
/fs{1 bt  
  2 bt  
  4 bt  
  8 bt  
 16 bt  
  sa astore pop  
}def
```

```
/mx1 matrix def  
/mx2 matrix def  
/gf(currentfont)def
```

```
-----  
-----
```

```
% Text munging procedures.  
% These have to do with string merging and text scaling and rotation.
```

```
-----  
-----
```

```
-----  
-----
```

```
% set relative center of rotation:  
% yintegerpart yfractionpart xintegerpart xfractionpart
```

```
-----  
-----  
t
```

```
/tc(32768 div add 3 1 roll 32768 div add 2t astore pop)def
```

```
-----  
-----
```

```
% set additional rotation parameters: justify flip rotation
```

```
% justify is:  
% 1 left  
% 2 center  
% 3 right  
% else full  
% flip is:  
% 0 none  
% 1 flip around y axis  
% 2 flip around x axis  
% rotation is in degrees counterclockwise
```

```
-----  
-----
```

```
/3a [0 0 0] def  
/2t 2 array def  
/tp(3a astore pop)def
```



```

/ee{}def

%-----
%-----
% start rotated text at current penlocation:
%-----
%-----

/tt{gsave currentpoint 2 copy 2t aload pop qa 2 copy x1 3a aload pop exch
dup 0 eq
{pop}{1 eq{-1 1}{1 -1}ifelse scale}ifelse rotate pop neg exch neg exch x1
moveto}def

/te{currentpoint currentfont grestore setfont moveto}def
/tb{/tg currentgray def 3 -1 roll 3 eq{1 setgray}if /ml 0 def /al 0 def}def
/am{ml add /ml xdf}def
/aa{{currentgray /setgray cvx}cvx exch dup wi pop dup al add /al xdf
exch}def

%-----
%-----
% scale coordinate system by numerator denominator pairs:
% denominator(vertical,horizontal) numerator(vertical,horizontal)
%-----
%-----

/th{3 -1 roll div 3 1 roll exch div 2 copy mx1 scale pop scale /scaleflag
true def}def
/tu{1 1 mx1 itransform scale /scaleflag false def}def
/ts{1 1 mx1 transform scale /scaleflag true def}def

%-----
%-----
% fontsize: pointsize
%-----
%-----

/fz{/ps xdf}def

%-----
%-----
% execute a procedure but leave procedure on stack
% {proc}fx
%-----
%-----

/fx{dup exec}def

/st{show pop pop}def

%-----
%-----

```

```

% These procedures constitute string merging.
%-----
/tn(
{
dup type dup /integertype eq exch /realttype eq or
{
dup ml mul
}
{
dup type /stringtype eq
{
rs
}
{
dup type /dicttype eq
{
setfont
}
{
dup type /arraytype eq
{
exec
}
{
pop
}ifelse
}ifelse
}ifelse
}ifelse
}forall
}def
%-----
% textFace textmode justification-type [array-of-strings-and-font-changes]
es
%-----
/es{
3 -1 roll dup sa 5 get dup type /nulltype eq
{pop4 pop}
{
sa 1 get (/ml ml .2 ps mul sub def)if %Italic Hack Hack Hack
ne(fs){pop}ifelse exch
dup 1 eq
%left justification
{pop
al ml gt(/tv(ll)/ml ml al dup 0 ne(div){pop}ifelse def){/tv(st)/ml
1 def}ifelse def
}
}

```

```

        tm
    }
    {
    dup 3 eq
%right justification
    {pop
    al ml gt{/tv{ll}/ml ml al dup 0 ne{div}{pop}ifelse def}{ml al sub 0
rmoveto /tv{st}/ml 1 def}ifelse def
        tm
    }
    {
    2 eq
%center justification
    {
    al ml gt{/tv{ll}/ml ml al dup 0 ne{div}{pop}ifelse def}{ml al sub 2
div 0 rmoveto /tv{st}/ml 1 def}ifelse def
        tm
    }
    {
%full justification
    /tv{ll}def
    /ml ml al dup 0 ne{div}{pop}ifelse def
        tm
    }ifelse}ifelse}ifelse
    }ifelse
    tg setgray
}def

```

```

/pop4 (pop pop pop pop) def

```

```

%-----
%-----
% The QuickDraw Procedures
%-----
%-----
%-----
% moveto: x y
%-----
%-----
/gm(scaleflag{mx1 itransform}if moveto)def
%local ymove: x y ly
/ly(exch pop currentpoint exch pop sub 0 exch rmoveto)def
%-----
%-----
% print n copies of page (ensures 8 pages/minute for multiple copies)
%-----
%-----

```

```

/page{1 add /#copies xdf showpage}def

/sk{systemdict /statusdict known}def

%-----
% set jobname (string)
%-----

/jn{sk{statusdict /jobname 3 -1 roll put}{pop}ifelse}def

%-----
% set pen size: h v pen
%-----

/pen {/pnsv xdf /pnsh xdf pnsh setlinewidth} def

%-----
% lineto procedures: x y
% (uses current pen location, pen size and graylevel)
% This really emulates the ugly QuickDraw pen on the LaserWriter but
preserves the same
% endpoint and linewidth anomalies that some applications rely on.
%-----

/dlin{currentpoint newpath moveto lineto currentpoint stroke grestore
moveto}def
/lin {currentpoint /pnlv xdf /pnlh xdf gsave newpath /@y xdf /@x xdf fp{pnlh
@x lt {pnlv @y ge
{pnlh pnlv moveto @x @y lineto pnsh 0 rlineto
0 pnsv rlineto pnlh pnsh add pnlv pnsv add lineto pnsh neg 0 rlineto}
{pnlh pnlv moveto pnsh 0 rlineto @x pnsh add @y lineto 0 pnsv rlineto
pnsh neg 0 rlineto pnlh pnlv pnsv add lineto} ifelse} {pnlv @y gt
{@x @y moveto pnsh 0 rlineto pnlh pnsh add pnlv lineto 0 pnsv rlineto
pnsh neg 0 rlineto @x @y pnsv add lineto} {pnlh pnlv moveto pnsh 0 rlineto
0 pnsv rlineto @x pnsh add @y pnsv add lineto pnsh neg 0 rlineto
0 pnsv neg rlineto} ifelse} ifelse
closepath fill)if @x @y grestore moveto} def
/dl{gsave 0 setlinewidth 0 setgray}def

%-----
% Arc: top left bottom right startangle stopangle verb flag
% flag means to exclude the center of curvature in the arc
%-----

/barc {/@f xdf /@op xdf /@e xdf /@s xdf /@r xdf

```

```

/@b xdf /@l xdf /@t xdf gsave
@r @l add 2 div @b @t add 2 div xl 0 0 moveto
@r @l sub @b @t sub mtx currentmatrix pop scale @f {newpath} if
0 0 0.5 @s @e arc
mtx setmatrix @op doop grestore) def
/doarc (dup 0 eq barc) def

%-----
% oval: top left bottom right verb
%-----

/doval (0 exch 360 exch true barc) def

%-----
% rectangle: top left bottom right verb
/dorect {/@op xdf currentpoint 6 2 roll newpath 4 copy
4 2 roll exch moveto 6 -1 roll lineto lineto lineto closepath
@op doop moveto} def
/mup{dup pnsv 2 div le exch pnsv 2 div le or}def

%-----
% roundrect: top left bottom right ovalwidth ovalheight operation
% Caveat: ovalwidth is currently assumed equal to ovalheight
%-----

/dorrect {/@op xdf 2. div /@h xdf 2. div /@w xdf
/@r xdf /@b xdf /@l xdf /@t xdf
@t @b eq @l @r eq @w mup or or(@t @l @b @r @op dorect)
{
  @r @l sub 2. div dup @w lt{/@w xdf}{pop}ifelse
  @b @t sub 2. div dup @w lt{/@w xdf}{pop}ifelse
  @op 0 eq{/@w @w pnsv 2 div sub def}if %this helps solve overlap gap
for wide line widths
  currentpoint
  newpath
  @r @l add 2. div @t moveto
  @r @t @r @b @w arcto pop4
  @r @b @l @b @w arcto pop4
  @l @b @l @t @w arcto pop4
  @l @t @r @t @w arcto pop4
  closepath @op doop
  moveto
}ifelse
} def

```

```

%-----
%-----
% Polygon utility procedures
%-----
%-----
/pr(gsave newpath /pl(moveto /pl(lineto)def)def)def
/pl{lineto}def
/ep{dup 0 eq
  {
    (moveto){lin}({})pathforall %nothing but movetos and linetos should be
called
    pop grestore
  }
  {
    doop grestore
  }ifelse
}def
/bs 8 string def
/bd{/bs xdf}def

%-----
%-----
% These following procedures are used in defining QuickDraw patterns.
% Pattern definition goes into halftone screen of PostScript
%-----
%-----
% procedure to find black bits in QuickDraw pattern (pattern in hex string
bs)
/bit (bs exch get exch 7 sub bitshift 1 and) def
/bix (1 add 4 mul cvi) def
/pp(exch bix exch bix bit)def
/grlevel (64. div setgray) def

%-----
%-----
% procedure to set a pattern: ratio hexstring
% ratio is the total number of white bits in the QuickDraw pattern
represented in hexstring
%-----
%-----
/setpat (/bs xdf 9.375 0(pp)setscreen grlevel) def
/setgry (freq rot (spf) setscreen grlevel) def

%-----
%-----
% standard copybits routine: xscale yscale xloc yloc rowbytes xwidth ywidth
fsmooth bitmode
% This procedure is the basis for all QuickDraw bit operations.
% xscale and yscale tell how much to scale the bit image in 72nds of an inch

```

```

% xloc and yloc are the location of the top left corner of the bitmap
% rowbytes is the total number of bytes in each scanline of hex data in the
image
% Note that rowbytes must be even
% xwidth and ywidth are the actual number of bits in the x and y coordinates
of the image
% fsmooth is a flag to tell whether or not to use bit smoothing. Bit
smoothing is a
% proprietary algorithm that provides smoothing of the data around a 5 by
5 local area of each
% data pixel.
% bitmode can be any of the QuickDraw source transfer modes excluding srcXor
and notSrcXor
% Note that this is the only QuickDraw procedure which can implement more
than the simple
% srcCopy transfer mode.
%-----
-----1
/x4 (2 bitshift) def
/d4 (-2 bitshift) def
/xf (.96 mul exch 2 sub .96 mul exch) def
/dobits
{
/bmode xdf
save 9 1 roll
% 2 sub fixes dxsrc offset number required for bitsmoothing, but applies to
both
%Bit Smooth mode
{
x4 /@dy xdf 2 sub x4 /@dx xdf /@idx xdf
.96 mul exch 3 index 2 sub @dx div 7.68 mul dup 6 1 roll sub exch x1 0 0
moveto xf
0 4 -1 roll 2 index 4 index 1.759 add 10 dorect clip newpath 0 0 moveto
scale
bmode 0 eq bmode 4 eq or{1 setgray 1 @dy div 1 @dx div 1 1 2 dorect}if
bmode 3 eq bmode 7 eq or{1}{0}ifelse setgray
@idx 5 bitshift @dy bmode 0 eq bmode 1 eq bmode 3 eq or or [@dx 0 0 @dy 0
0]
{(%stdin)(r) file @dy d4 4 add @idx mul string readhexstring pop
dup length @idx x4 sub 4 bitshift string
dup 3 1 roll @dx 8 add d4 smooth} imagemask
}
}
%Non Bit Smooth mode
{
/@dy xdf 2 sub /@dx xdf /@idx xdf
/@xs @idx string def
/@f (%stdin)(r) file def
/@p{@f @xs readhexstring pop}def
.96 mul x1 0 0 moveto xf scale
}

```

```

0 0 1 1 10 dorect clip newpath 0 0 moveto
bmode 0 eq bmode 4 eq or{1 setgray .25 @dy div .25 @dx div 1 1 2
dorect}if
bmode 3 eq bmode 7 eq or{1}{0}ifelse setgray
@p @p
@idx 3 bitshift @dy bmode 0 eq bmode 1 eq bmode 3 eq or or [@dx 0 0 @dy 0
0]
(@p) imagemask
@p @p pop4
}ifelse
restore
} def

```

```

%-----
% Making Mac compatible Fonts
%-----

```

```

/mfont 14 dict def
/wd 14 dict def
/mdef {mfont wcheck not{/mfont 14 dict def}if mfont begin xdf end} def
/dc {transform round .5 sub exch round .5 sub exch itransform} def

```

```

%-----
% Copy a font dictionary: fontdictionary
% copies a font dictionary into tmp so it may be used to define a new font
%-----
% tmp must be set before cf is called
/cf{{1 index /FID ne {tmp 3 1 roll put}{pop pop}ifelse}forall}def

```

```

%-----
% Procedures used in defining a bit map font
%-----

```

```

/mv{tmp /Encoding macvec put}def
/bf{
mfont begin
/FontType 3 def
/FontMatrix [1 0 0 1 0 0] def
/FontBBox [0 0 1 1] def
/Encoding macvec def
/BuildChar
{
wd begin
/cr xdf

```



```

/fd xdf
fd /low get cr get 2 get -1 ne
{
fd begin
  low cr get aload pop
  sd
  low cr 1 add get 0 get
  sh
  sw
end
/sw xdf
/sh xdf
sw div /clocn xdf
dup 0 ne {0 exch sh div neg dc xl}{pop}ifelse
exch sw div /coff xdf
exch sw div /cloc xdf
/bitw clocn cloc sub def
sw sh div 1 scale
sw div 0 coff 0 bitw coff add 1 setcachedevice
coff cloc sub 0 dc xl
cloc .5 sw div add 0 dc newpath moveto
bitw 0 ne
  {0 1 rlineto bitw .5 sw div sub 0 rlineto 0 -1 rlineto
  closepath clip
  sw sh false [sw 0 0 sh neg 0 sh]{fd /hm get}imagemask}if
} if
end
} def
end
mfont definefont pop
} def

```

```

%-----

```

```

% stringwidth procedure which does not allow a show to occur: (string)
%-----

```

```

/wi{save exch /show(pop)def
stringwidth 3 -1 roll restore}def

```

```

/aps {0 get 124 eq}def
/apn {s30 cvs aps} def

```

```

%-----

```

```

%set style in a PostScript name: AppleFontName
% e.g.
% /|----name sos /|---Oname
% /|----name sis /|-I---name

```

%-----

```
-----  
/xc(s30 cvs dup)def  
/xp(put cvn)def  
/scs(xc 3 67 put dup 0 95 xp)def  
/sos(xc 3 79 xp)def  
/sbs(xc 1 66 xp)def  
/sis(xc 2 73 xp)def  
/sob(xc 2 79 xp)def  
/sss(xc 4 83 xp)def  
  
/dd(exch 1 index add 3 1 roll add exch) def  
/smc(moveto dup show) def  
/kwn(dup FontDirectory exch known(findfont exch pop))def  
/fb(/ps ps 1 add def)def  
/mb  
{dup sbs kwn  
  {  
    exch(pop){bbc}{} mm  
  }ifelse  
sfd  
}def  
/mo  
{dup sos kwn  
  {  
    exch(pop){boc}{} mm  
  }ifelse  
sfd  
}def  
/ms  
{dup sss kwn  
  {  
    exch(pop){bsc}{} mm  
  }ifelse  
sfd  
}def  
/ao  
{dup sos kwn  
  {  
    exch dup ac pop  
    {scs findfont /df2 xdf}{aoc}{} mm  
  }ifelse  
sfd  
}def  
/as  
{dup sss kwn  
  {  
    exch dup ac pop  
    {scs findfont /df2 xdf}{asc}{} mm
```

```

)ifelse
sfd
)def
/ac
{
  dup scs kwn
  {exch /ofd exch findfont def
  /tmp ofd maxlength 1 add dict def
  ofd cf mv
  tmp /PaintType 1 put
  tmp definefont}ifelse
}def
/mm{
/mfont 10 dict def
mfont begin
/FontMatrix [1 0 0 1 0 0] def
/FontType 3 def
/Encoding macvec def
/df 4 index findfont def
/FontBBox [0 0 1 1] def
/xda xdf
/mbc xdf
/BuildChar { wd begin
  /cr xdf
  /fd xdf
  /cs sl dup 0 cr put def
  fd /mbc get exec
  end
} def
exec
end
mfont definefont} def
/bbc
{
  /da .03 def
  fd /df get setfont
  gsave
  cs wi exch da add exchd
  grestore
  setcharwidth
  cs 0 0 smc
  da 0 smc
  da da smc
  0 da moveto show
} def
/boc
{
  /da 1 ps div def
  fd /df get setfont

```

```

gsave
  cs wi
  exch da add exch
grestore
setcharwidth
cs 0 0 smc
  da 0 smc
  da da smc
  0 da smc
1 setgray
  da 2. div dup moveto show
} def
/bsc
{
  /da 1 ps div def
  /ds .05 def %da dup .03 lt {pop .03}if def
  /da2 da 2. div def
  fd /df get setfont
  gsave
    cs wi
    exch ds add da2 add exch
  grestore
  setcharwidth
  cs ds da2 add .01 add 0 smc
    0 ds da2 sub xl
    0 0 smc
  da 0 smc
  da da smc
  0 da smc
  1 setgray
    da 2. div dup moveto show
} def
/aoc
{
  fd /df get setfont
  gsave
    cs wi
  grestore
  setcharwidth
  1 setgray
  cs 0 0 smc
  fd /df2 get setfont
  0 setgray
  0 0 moveto show
} def
/asc
{
  /da .05 def
  fd /df get setfont

```

```

gsave
  cs wi
  exch da add exch
grestore
setcharwidth
cs da .01 add 0 smc
  0 da xl
  1 setgray
  0 0 smc
  0 setgray
  fd /df2 get setfont
  0 0 moveto show
)def

```

```

%-----
%-----
% Procedure to print instruction sheet and set up manual feed
%-----
%-----

```

```

/mf(gsave
32 760 xl 1 -1 scale
1 1 pen
128 152 moveto
27.5 27.5 693.5 522.5 0 dorect
6 6 pen
63. 63. 657. 486. 0 dorect
48 fz F /|B---1Times fnt pop
(Manual Feed)show
118 275 moveto
14 fz F /|----1Times fnt pop
(document: )show
sk(statusdict /jobname get dup null ne(show){pop}ifelse)if
%("")show
118 362 moveto
(Manual Feed Instructions)show
127 398 moveto
(1. Wait until the yellow light on the front of your)show
145 416 moveto
(LaserWriter comes on steadily \5C(not flashing\5C).)show
127 458 moveto
(2. Insert your paper or envelope in the manual feed)show
145 478 moveto
(guide on the right side of the LaserWriter.)show
127 517 moveto
(3. Repeat steps 1 and 2 until your document is)show
145 537 moveto
(completed.)show
0 page
sk(statusdict /manualfeed true put 5 dly)if

```

```

grestore)def
/dly{
usertime exch 1000 mul add
{
dup usertime le(exit)if
}loop
pop
}def

%-----
% List all Apple compatible fonts one name per line
%-----

/lsf {FontDirectory {pop dup apn(= flush){pop}ifelse}forall /* = flush}def

/T true def
/F false def

%-----
% More Polygon stuff used in polygon comment
%-----

/6a 6 array def
/2a 2 array def
/5a 5 array def
%subtract points, first from second (reverse order): pt0 pt1 qs newpt
/qs{3 -1 roll sub exch 3 -1 roll sub exch}def
/qa{3 -1 roll add exch 3 -1 roll add exch}def
%multiply point: pt factor qm newpt
/qm{3 -1 roll 1 index mul 3 1 roll mul}def
/qn{6a exch get mul}def
/qA .166667 def /qB .833333 def /qC .5 def
/qx{
6a astore pop
qA 0 qn qB 2 qn add qA 1 qn qB 3 qn add
qB 2 qn qA 4 qn add qB 3 qn qA 5 qn add
qC 2 qn qC 4 qn add qC 3 qn qC 5 qn add
}def
/qp{6 copy 12 -2 roll pop pop}def
/qc{qp qx curveto}def
/qi{{4 copy 2a astore aload pop qa .5 qm newpath moveto}{2 copy 6 -2 roll 2
qm qs 4 2 roll}ifelse}def
/qq{{qc 2a aload pop qx curveto}{4 copy qs qa qx curveto}ifelse}def
%start polygon comment
/pt{gsave currentpoint newpath moveto}def
%fill smoothed poly
/qf{gsave eofill grestore}def

```

```

/tr{currentgray currentscreen bs 5a astore pop /fillflag 1 def}def
/bc{/fillflag 0 def}def
%polyverb ec
/ec{currentpoint 3 -1 roll
  1 and 0 ne
  (currentgray currentscreen bs 5a aload pop bd setscreen setgray 0 doop bd
setscreen setgray)
  (newpath)ifelse
  moveto
}def
/bp{currentpoint newpath 2 copy moveto currentgray currentscreen bs 5a
astore pop}def
/eu{
  fillflag 0 ne
  (
  gsave currentgray currentscreen bs
  5a aload pop bd setscreen setgray
  4 ep
  bd setscreen setgray
  )if
  fp(0 ep){grestore newpath}ifelse
}def

%-----
%-----
% Line Layout stuff used by string merging algorithm
%-----
%-----
% counts spaces in string:  (...) sm (...) n
% returns string and number of spaces in string
%-----
%-----
/sm
{
dup 0 exch
{32 eq{1 add}if}forall
}
def

%-----
%-----
% layout a string to length specified by desiredlength:  printerlength
desiredlength (...) ll
% printerlength is length of string in printer space
%-----
%-----
/ll
{
3 1 roll exch dup .0001 lt 1 index -.0001 gt and

```

```

(pop pop pop)
(sub dup 0 eq
  {
    pop show
  }
  {
    1 index sm dup 0 eq 3 index 0 le or
    {
      pop length div
      0 3 -1 roll ashow
    }
    {
% This piece does 10 percent stretching in characters and 90 percent in
spaces
      10 mul exch length add div
      dup 10 mul 0 32 4 -1 roll 0 6 -1 roll awidthshow
% This piece does straight stretching in spaces only
%   exch pop div
%   0 32 4 -1 roll widthshow
    }ifelse
  }ifelse
}ifelse
}def

%-----
%-----
%set font to symbol and show the string: (...) ss
%-----
%-----
/ss
{ /pft currentfont def sa aload pop pop /|----2Symbol 4 1 roll
  {pop{as}}
  {{{ao}}{fnt}}ifelse}ifelse
  exch pop exec exch pop
}def
/pf{pft dup setfont}def

%-----
%-----
% regular show does underline if ulf is true: printerlength desiredlength
string rs
%-----
%-----
/rs
{
  sa 2 get
  {
    gsave
    1 index 0

```



```

currentfont
dup /FontInfo known
{
  /FontInfo get
  dup /UnderlinePosition known
  {
    dup /UnderlinePosition get 1000 div ps mul
  }
  {
    ps 10 div neg %15 makes line closer to text
  }ifelse
exch
dup /UnderlineThickness known
{
  /UnderlineThickness get 1000 div ps mul
}
{
  pop
  ps 15 div %20 makes slightly narrower line
}ifelse
}
{
  pop
  ps 10 div neg %15 makes line closer to text
  ps 15 div %20 makes slightly narrower line
}ifelse
setlinewidth
0 setgray
currentpoint 3 -1 roll sub moveto
sa 4 get(gsave currentlinewidth 2. div dup rmoveto currentpoint x1 2 copy
rlineto
stroke grestore)if
sa 3 get sa 4 get or 3 1 roll 2 index(gsave 1 setgray 2 copy rlineto
stroke grestore)if
rlineto(strokepath 0 setlinewidth)if stroke
grestore
}if
tv
}
def

%-----
%
% More Font building stuff, specifically the Apple Encoding Vector
%-----

% Font encoding vector for PostScript fonts to match Mac
/macvec 256 array def
macvec 0

```

```

        (Bold) search (/@b true def) if (Roman) search pop (-) search pop /@s
xdf cleartomark
    @s cvn dup /Symbol eq(pop 50){/Courier eq(51){49}ifelse}ifelse
s30 0 @s length 6 add getinterval dup 6 @s putinterval dup 0 (|-----)
putinterval
    @b {dup 1 66 put} if @i @o or {dup 2 73 put} if % @o {dup 2 79 put} if
    dup 5 4 -1 roll put
    cvn tmp definefont pop
    }ifelse
}forall

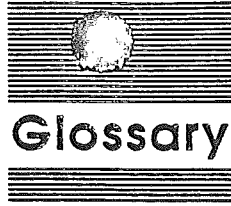
%-----
%-----
%Make any other special fonts here, i.e. Seattle
%-----
%-----
/_--C-2Symbol /Symbol findfont /tmp 1 index maxlength 1 add dict def cf tmp
/PaintType 1 put tmp definefont
/|----4Seattle /Helvetica findfont dup length 1 add dict /tmp xdf cf mv
/mxv [/zero /one /two /three /four /five /six /seven /eight /nine /comma
/period /dollar /numbersign
/percent /plus /hyphen /E /parenleft /parenright /space] def
tmp /Metrics 21 dict dup begin mxv(600 def)forall end put
tmp begin /FontBBox FontBBox [0 0 0 0] astore def end
tmp definefont pop

%-----
%-----
% open document, open page and close page procedures
% close document doesn't do anything currently
%-----
%-----
% txpose takes the vertical page size as a parameter
/od{txpose 10 fz 0 fs F /|----3Courier fnt pop}def
/op{/scaleflag false def /pm save def}def
/cp{pm restore}def

end

```





To be provided: .